SCALABLE MODEL-BASED REINFORCEMENT LEARNING IN COMPLEX, HETEROGENEOUS ENVIRONMENTS

NGUYEN THANH TRUNG

B.Sci. in Information Technology

Ho Chi Minh City University of Science

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2013

Acknowledgements

I would like to thank:

Professor Leong Tze Yun, my thesis supervisor, for her guidance, encouragement, and support throughout my PhD study. I would not have made it through without her patience and belief in me.

Dr. Tomi Silander, my collaborator and mentor, for teaching me about effective presentation of technical ideas, and for the numerous hours of invaluable discussions. He has been a great teacher and a best friend.

Professor David Hsu and Professor Lee Wee Sun for reading my thesis proposal and providing constructive feedback to refine my work. Professor David Hsu together with Professor Leong Tze Yun have also offered me a research assistantship to work and learn in one of their ambitious collaborative projects.

Professor Tan Chew Lim and Professor Wynne Hsu for reading my graduate research proposal and for suggesting me helpful papers supporting my early research.

Mr. Philip Tan Boon Yew at MIT Game Lab and Mrs. Teo Chor Guan at Singapore-MIT GAMBIT Game Lab for providing me a wonderful opportunity to experience MIT culture.

Dr. Yang Haiqin at the Chinese University of Hong Kong for his valuable discussion and comments on Group Lasso, an important technical concept used in my work.

Members of the Medical Computing Research Group at the School of Computing, for their friendship and for their efforts in introducing interesting research ideas to the group in which I am a part.

All my friends who have helped and brightend my life over the years at NUS, especially Chu Duc Hiep, Dinh Thien Anh, Le Thuy Ngoc, Leong Wai Kay, Li Zhuoru, Phung Minh Tan, Tran Quoc Trung, Vo Hoang Tam, Vu Viet Cuong.

My grandmother, my parents for their unbounded love and encouragement. My brother and sister for their constant support. My uncle's family, Nguyen Xuan Tu, for taking care of me many years in my undergraduate study.

My girl friend, Vu Nguyen Nhan Ai, for sharing the joy and the sorrow with me, for her patience and belief in me, and most importantly for her endless love.

This research was supported by a Research Scholarship, and two Academic Research Grants: MOE2010-T2-2-071 and T1 251RES1005 from the Ministry of Education in Singapore.

Table of Contents

Summary

A system that can automatically learn and act based on feedback from the world has many important applications. For example, the system may replace humans to explore dangerous environments such as Mars, the ocean, or to allocate resources in an information network, or to drive a car home without requiring a programmer to manually specify rules on how to do so. At this time the theoretical framework provided by reinforcement learning (RL) appears quite promising for building such the system.

There has been a large number of studies focusing on RL to solve challenging problems. However, in complex environments, much domain knowledge is usually required to carefully design a small feature set to control the problem complexity; otherwise, it is almost likely computationally infeasible to solve the RL problems with the state of the art techniques. An appropriate representation of the world dynamics is essential to efficient problem solving. Compactly represented world dynamics models should also be transferable between tasks, which may then further improve the usefulness and performance of the autonomous system.

In this dissertation, we first propose a scalable method for learning the world dynamics of feature-rich environments in model-based RL. The main idea is formalized as a new, factored state-transition representation that supports efficient online-learning of the relevant features. We construct the transition models through predicting how the actions change the world. We introduce an online sparse coding learning technique for feature selection in high-dimensional spaces.

Second, we study how to automatically select and adapt multiple abstractions or representations of the world to support model-based RL. We address the challenges of transfer learning in heterogeneous environments with varying tasks. We present an efficient, online method that, through a sequence of tasks, learns a set of relevant representations to be used in future tasks. Without pre-defined mapping strategies, we introduce a general approach to support transfer learning across different state spaces. We demonstrate the jumpstart and faster convergence to near optimum effects of our system.

Finally, we implement these techniques in a mobile robot to demonstrate their practicality. We show that the robot equipped with the proposed learning system is able to learn, accumulate, and transfer knowledge in real environments to quickly solve a task.

Publications from the dissertation research work

1. Online Feature Selection for Model-based Reinforcement Learning,
   Trung Thanh Nguyen, Zhuoru Li, Tomi Silander, Tze-Yun Leong,
   Proceedings of the International Conference on Machine Learning (ICML '13),
   Atlanta, USA, June 2013.

   We propose a new framework for learning the world dynamics of feature-rich environments in model-based reinforcement learning. The main idea is formalized as a new, factored state-transition representation that supports efficient online-learning of the relevant features. We construct the transition models through predicting how the actions change the world. We introduce an online sparse coding learning technique for feature selection in high-dimensional spaces. We derive theoretical guarantees for our framework and empirically demonstrate its practicality in both simulated and real robotics domains.

2. Transferring Expectations in Model-based Reinforcement Learning,
   Trung Thanh Nguyen, Tomi Silander, Tze-Yun Leong,
   Proceedings of the Advances in Neural Information Processing Systems (NIPS'12),
   Lake Tahoe, Nevada, USA, December 2012.

   We study how to automatically select and adapt multiple abstractions or representations of the world to support model-based reinforcement learning. We address the challenges of transfer learning in heterogeneous environments with varying tasks. We present an efficient, online framework that, through a sequence of tasks, learns

a set of relevant representations to be used in future tasks. Without pre-defined mapping strategies, we introduce a general approach to support transfer learning across different state spaces. We demonstrate the potential impact of our system through improved jumpstart and faster convergence to near optimum policy in two benchmark domains.

3. Transfer Learning as Representation Selection,
   Trung Thanh Nguyen, Tomi Silander, Tze-Yun Leong,
   International Conference on Machine Learning Workshop on Representation Learning (ICML'12), Edinburgh, Scotland, June 2012.

   An appropriate representation of the environment is often key to efficient problem solving. Consequently, it may be helpful for an agent to use different representations in different environments. In this paper, we study selecting and adapting multiple abstractions or representations of environments in reinforcement learning. We address the challenges of transfer learning in heterogeneous environments with varying tasks. We present a system that, through a sequence of tasks, learns a set of world representations to be used in future tasks. We demonstrate the jumpstart and faster convergence to near optimum effects of our system. We also discuss several important variants of our system and highlight assumptions under which these variants should improve the current system.

List of Tables

List of Figures

Chapter 1

Introduction

"Dirt roads in my village were sandwiched between rice fields, which was always challenging to riders due to the uneven terrain. The roads were narrow, and covered with rocks and grass on both sides. Carelessly riding on the sides of the roads would easily send a bicycle off. Sometimes, the bicycle would skid, and get stuck in the nearby rice fields. After years of secondary school, I was sent to a city town for high school. Though the town was just 15 kilometers away, its terrain was new to me. Roads were wider, and made of tarmac more solid than the dried mixture of mud and soil in my rural village. Roadsides were filled with, instead of colorful fields, houses and shops. The bicycle, though turned unexpectedly on the pavements, could move in my desired direction most of the time. Years later, I went to the capital city, and now, Singapore – their road systems may be better but the basic characteristics and dynamics are quite *similar to my experience in my high school town*. Likewise, summer riding on roads in England is quite the same, but I would *expect* it to be much different in winter time like snowy winters in Tokyo, when street lights may not be enough especially in urban areas; roads are wet; road markings tend to be slippery, as do drain and manhole covers. A sharp turn over a wet piece of iron work or painted

line at full speed could easily result in a fall." – Here is a bicycling story based on my own experiences. It describes a common activity in our daily lives.

1.1 Motivations

Real environments are complex – they contain large numbers of different pieces of information or features that may or may not affect the outcomes of one's actions. In practice, an analysis upon all these features would carry prohibitive costs preventing action decisions to be made on time. Instead of analyzing every aspect of the situation, humans are able to operate efficiently in these environments due to their capability of focusing attention on just a few key features to capture the world dynamics. For example, we see that, in the bicycling story, rocks and grass, but not the colors of rice, flowers on roadsides or any others, are reasons that make a bicycle skid in the rural village; or that road markings, drain, and manhole covers tend to be slippery in snowy winter in England and Tokyo. It appears that based on feedback of their interactions with an environment, humans select features to form models or *views* to approximate the world dynamics. A view is a way to "look" at the world.

Humans seem to also accumulate knowledge during their life time to increase adaptability in an environment. While riding a bicycle in a rural village, in Saigon city in Vietnam, in Tokyo in winter, etc. the rider forms different views and carries on to his riding in Singapore, and England. The views reflect the rider's different expectations about the world dynamics in a new place. By suitable expectations, the rider may quickly capture the dynamics to operate efficiently. While some of these capabilities of humans may well be innate, artificial intelligence agents without evolutionary traits may have to resort to machine learning (Bishop 2006).

Building an autonomous agent that could, like humans, learn and act by feedback from environments is an important goal in artificial intelligence research. Due to its promise of freeing programmers from the challenging tasks of specifying rules for

the agent to act, reinforcement learning (RL) has been recently a popular approach (Kaelbling, Littman, and Moore 1996). In RL, an agent's task is framed as a sequential decision making problem in which after each action, an agent will receive feedback from the environment for its decision. The feedback informs, for example, that riding forward from the last position has taken the bicycle forward, or that it has thrown the bicycle to the rice fields. Feedback can also be positive or negative signals such as falling down on the way. An RL agent then uses this feedback to capture the dynamics of the environment, and to plan its actions automatically. The dynamics of an environment, or the *world dynamics* is the source that determines the outcomes of an agent's action at each situation in an environment. In other words, it determines the feedback for each agent's action.

An RL problem is typically modeled in a Markov decision process (MDP) (Sutton and Barto 1998). A task has a set of states. An agent performs actions to transit from one state to another state aiming to have the highest positive feedback signals or rewards. The world dynamics is modeled through functions of states and actions. In large environments, states are usually factored, and the dynamics is represented by dynamic Bayesian networks (DBN) to capture the structures underlying the world dynamics (Kearns and Koller 1999). Hopefully, knowledge could be generalized efficiently without requiring the agent to visit every state. However, learning DBNs to represent the dynamics of a complex environment is difficult and often computationally infeasible. By imposing different assumptions, numerous methods have been proposed (Hester and Stone 2009; 2012; Diuk, Li, and Leffler 2009; Chakraborty and Stone 2011). Several transfer learning techniques have also been suggested to accelerate the learning (Atkeson, Moore, and Schaal 1997; Wilson et al. 2007; Fernández, García, and Veloso 2010). The state of the art methods, however, are not scalable to complex, feature-rich environments. Working in *heterogeneous environments* is yet another challenge. In heterogeneous settings, the world dynamics, feature distributions, state spaces, or terminal states in different environments may be very different.

1.2 Research problems

Focusing on model-based RL, this dissertation examines the problems of learning in complex environments. In particular, we focus on two problems: learning representations, and transferring representations. We limit the research to domains with discrete state and action spaces. Tasks are episodic. Environments are stationary; the dynamics of a stationary environment does not change over time.

1.2.1 Representation learning in complex environments

In order to operate in an environment, an autonomous agent has to be equipped with sensors to "see" the environment. The number of sensors may range from just a few to hundreds. The agent uses those sources of information to form features to describe the environment and to capture feedback for each of its actions. In practice, important features to approximate the dynamics of an environment are unknown. An autonomous agent might prepare a set of many features, which also possibly contains various redundant or irrelevant features, and rely on learning methods to gradually select important ones to represent the approximate dynamics model. However, the state of the art methods do not scale up to work with large feature vectors and big data. A few potentially important features have to be selected manually and encoded to the agent. Although this approach is possible in some applications, the "heavy" work is left for humans, which raises the question of autonomy of an artificial agent.

1.2.2 Representation transferring in heterogeneous environments

An RL method usually takes a long running time, and its result is specific for a task. Therefore, studies have concentrated on transfer learning methods which target on reusing knowledge learned in one task in another task. While there has been some progress (Taylor and Stone 2009), current methods often require many strong assumptions which can hardly be satisfied in practice. The problem is challenging

because during its life time an agent may experience tasks in various environments which may have different dynamics. In a new task, it is difficult to know which pieces of knowledge are useful for quickly approximating the dynamics of the new environment; applying experience in wrong places, or having a wrong expectation of the world dynamics may easily result in big losses. For instance, in England winter, one should use experience of riding in Tokyo streets instead of in Singapore or a rural village in southern Vietnam where snowy winters never occur.

1.3 Research objectives and approaches

We aim to build a life-long learning agent that could automatically and efficiently learn, and transfer knowledge over any tasks based solely on feedback from the environments. Within the scope described above, this work tries to answer the following questions:

- Provided that environments are complex and feature-rich in which many features are redundant or irrelevant to represent the agent's action outcomes, is there a simple and scalable way to model the world dynamics?
- How can those models/representations be learnt incrementally online to integrate into the model-based RL framework? In other words, how possible is it to implement the "attention focus" for model-based RL?
- Transfer learning can have both boosting and "hurting" effects on the performance of an autonomous agent. Given that environments are heterogeneous, how can we effectively reuse knowledge to learn the world dynamics of an environment?
- Can the strengths of the two above methods be integrated for a unified learning framework that enables a model-based RL agent to learn, accumulate, and transfer knowledge in every task?

1.3.1 Online feature selection

We propose a new method for learning the world dynamics of feature-rich environments in model-based RL. Based on the action effect concept in situation calculus (McCarthy 1963) and a new principal way to distinguish the roles of features, we introduce a customized DBN to model the world dynamics. We show a sparse multinomial logistic regression algorithm that effectively selects relevant features and learns the DBN online.

1.3.2 Transfer learning in heterogeneous environments

We study how to automatically select and adapt multiple abstractions or representations of the world to support model-based RL. We address the challenges of transfer learning in heterogeneous environments with varying tasks. We present an efficient, online method that, through a sequence of tasks, learns a set of relevant views/representations to be used in future tasks.

1.3.3 Empirical evaluations in a real robotic domain

In RL, the theoretical results are usually defined under several simplified assumptions such as that the world dynamics could be modeled by certain distribution families, or that feedback data are independently and identically observed. Therefore, it is not clear if the results directly translate to enhance performance of an autonomous agent in real world domains.

To understand the practical quality of our theoretical framework, we will also conduct experiments on a robotic domain. We examine if our feature selection algorithm enables an agent to work efficiently and whether our framework of transferring views can significantly improve performance of an autonomous agent by knowledge accumulated over tasks.

1.4 Contributions

This work has the following main contributions:

Firstly, a variant formulation of the factored MDP that incorporates a principled way to compactly factorize the state space, while capturing comprehensive world dynamics information is proposed. This formulation establishes a uniform model dynamics representation to support RL in varying tasks and heterogeneous environments, and lowers the computational costs for structure learning in combinatorial spaces. We also provide an online multinomial logistic regression method with group lasso to learn the dynamics models/representations. Regret bound of the algorithm is also proved.

Secondly, a model-based RL with "attention focus" or online feature selection capability is presented. We show how to implement a model-based RL based on our variant MDP formulation. The algorithm performance is theoretically and empirically demonstrated.

Thirdly, a multi-view or multi-representation transfer learning approach is introduced. Without pre-defined mapping strategies, we show a general approach to support transfer learning across different state spaces, and with possibly different dynamics. We also develop a unified learning framework, which is a combination of our proposed transfer learning method and the new model-based RL algorithm above. As a result, it is possible to build an intelligent agent that automatically learns, and transfers knowledge to "progress" in its life time.

Finally, this dissertation includes a practical analysis of the proposed methods. We are interested in putting the system into real applications. Towards this end, we evaluate and discuss the strengths and weaknesses of our approach in two case studies in a robotic domain.

1.5 Report overview

This introductory chapter has briefly summarized the motivations and objectives of this research. The expected contributions have also been outlined. The subsequent chapters of the dissertation are organized as follows:

Chapter 2 reviews some background knowledge which we will need later in our method discussions. To keep the presentation concise, some detailed explanations will be referred to the appendices. This chapter also introduces current approaches to the two major problems: representation learning and transfer learning in model-based RL as described previously.

Chapter 3 presents an overview of our unified framework. It lays out key steps that will be addressed in the three following Chapters: 4, 5, and 6.

Chapter 4 describes our variant formulation of factored MDP. An online dynamics structure learning method and its analysis will also be introduced in this chapter.

Chapter 5 discusses a model-based RL based on our new MDP formulation introduced in Chapter 4. The chapter also includes theoretical and empirical results demonstrating how the RL agent can learn feature selection online to represent the world dynamics, and outperform the state of the art methods.

Chapter 6 introduces our representation transfer learning method. A detailed implementation of the unified learning framework is presented in this chapter. We also document empirical results demonstrating potential impact of the framework on the performance of an autonomous agent.

Chapter 7 examines the application of the proposed theoretical framework in a real robotic domain.

Chapter 8 summarizes the achievements as well as limitations of this work, and discusses future research.

For brevity, all the proofs are placed in the appendices.

Chapter 2

Background

This chapter first briefly reviews background knowledge for this work, and then continues with a survey of the current approaches to the research problems considered in this dissertation: representation learning and transfer learning in model-based RL.

2.1 Reinforcement learning

Reinforcement learning (RL), or learning by reinforcement signals, is a popular model for an autonomous agent to learn automatically and to operate in a stochastic environment (Sutton and Barto 1998). In RL, an agent performs actions to change its situations or states in the environment, aiming to maximize a numerical reward signal from the environment. The agent is not programmed with which actions to take at a situation, but has to interact with the environment to find out how to reach a goal. Figure 2-1 intuitively captures the interaction mechanism in RL. With a full or partial observation of an environment, the agent represents its situations in an environment as states in a state space. Upon performing an action, the agent will receive an immediate

reward from the environment. In addition, it will perceive a state transition. A new state may make subsequent rewards different.

These two types of feedback are critical for the agent to discover an action plan which can earn it the highest cumulative reward. In RL, any goal is transformed to a source of rewards. The problem of learning actions by trial-and-error search and delayed rewards differentiates RL from other methods.

An RL problem is typically formulated in terms of optimal control of MDPs, which will be described next. A Markov decision process imposes several assumptions on a learning task, but many studies on various environments, even in those where the assumptions are not fulfilled, have yielded successful results.

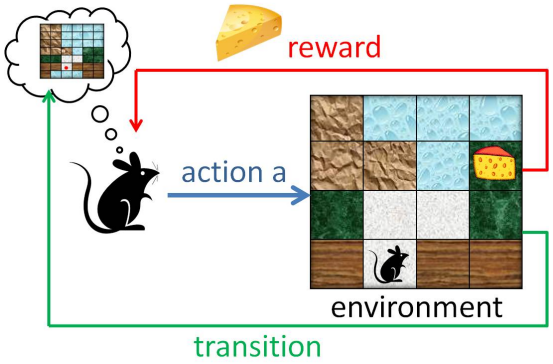


Figure 2-1: Reinforcement learning framework.

2.1.1 Markov decision process

A Markov decision process (MDP) is a 5-tuple (S, A, T, R, γ), where S is a set of states; A is a set of actions; $T : S \times A \times S \rightarrow [0, 1]$ is a stochastic transition function describing the dynamics of the environment, such that $T(s, a, s') = P(s'|s, a)$ indicates the probability of transiting to a state s' upon taking an action a at a state s; $R : S \times A \rightarrow \mathbb{R}$ is a reward function indicating expected immediate reward after an action a at a state s. An assumption while modeling a learning task by MDP is that the probability of moving to a new state by an action is conditionally independent of

the past states and actions given the current state, i.e., $P(s^{t+1}|s^t, a^t, s^{t-1}, a^{t-1}, .., s^0, a^0) = P(s^{t+1}|s^t, a^t)$.

Given an MDP, the goal is, then, to find an action policy $\pi : S \rightarrow A$ that specifies an action a to perform at each state s so that the expected cumulative future reward when starting from s is maximized. A policy can also be formulated non-deterministically, defining the probability that each action should be performed at a state (Sutton and Barto 1998), but we do not consider stochastic policies in this study. Rewards that occur t time steps in future are discounted by $\gamma^t \in (0; 1]$. The discount factor γ^t dictates that rewards received t time steps in the future are worth only γ^t times what it would be worth if it were received immediately. In case of infinite horizon planning in which the agent-environment interactions goes on infinitely, discount factors are specifically important relating to the availability of such a policy. This study, however, focuses on another popular case where an agent will stop at a special state called terminal state and will start the task again at another state. These tasks are called episodic tasks. In addition, we concentrate on finite MDPs in which the state and action spaces are finite. According to Sutton and Barto (1998), finite MDPs contribute to 90% of modern RL.

2.1.2 Value function and optimal policies

Given a policy, utility value of a state denotes expected future rewards for being at the state and following the policy thereafter. Let $V^\pi : S \rightarrow \mathbb{R}$ be a *value function* for a policy π. A state value, $V^\pi(s)$, is formally defined as:

$$V^\pi(s) = \mathbf{E}_T \left[\sum_{t=0}^{\mathsf{C}} \gamma^t R(s^t, \pi(s^t)) | \pi, s^0 = s \right], \tag{2.1}$$

where $\mathbf{E}_T$ denotes the expectation over the transition dynamics of the environment. The sum of future rewards is taken to $\mathsf{C} = \infty$ if the planning task has infinite horizon. For episodic tasks, where the agent will start over again when arriving at a terminal state, $\mathsf{C} \geq 0$ indicates the episode endings. s^t is a state where the agent is after

performing t actions according to π from s.

Similarly, value of a state given an action, called Q-value, is defined as below. The function defining Q-values is named Q-function to distinguish it from a value function.

$$Q^{\pi} : S \times A \rightarrow \mathbb{R} : Q^{\pi}(s,a) \quad = \quad \mathbf{E}_T \left[\sum_{t=0}^{\mathsf{C}} \gamma^t R(s^t, a^t) | \pi, s^0 = s, a^0 = a \right],$$

where $a^t = \pi(s^t)$ for every $t > 0$. $Q^{\pi}(s,a)$ measures expected future rewards for taking action a at state s and following the policy π thereafter.

A policy π^* is called optimal if $\forall \pi$, $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all states $s \in S$, i.e.,

$$\forall s \in S, V^{\pi^*}(s) = max_{\pi} V^{\pi}(s).$$

The value function for an optimal policy is called an optimal value function. Though there may be more than one optimal policies, they all share the same optimal value function (Sutton and Barto 1998). Let V^* be the optimal value function, then:

$$\forall s \in S, V^*(s) = max_{\pi} V^{\pi}(s).$$

Likewise, the optimal Q-function in an MDP, denoted Q^*, is also unique:

$$\forall s \in S, \forall a \in A, Q^*(s,a) = max_{\pi} Q^{\pi}(s,a).$$

Consequently, we can obtain that $V^*(s) = max_{a \in A} Q^*(s,a)$. Hence, an optimal policy can also be defined conveniently as:

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a). \tag{2.2}$$

In order to find an optimal policy π^*, most planning algorithms exploit the following fundamental property of value functions, which states a consistency condition

holding between a state and its successor states.

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^{\pi}(s'). \tag{2.3}$$

Equation 2.3 is well-known as *Bellman residual equation*, in which value of a state s is showed in a recursive relation to other state values, and has explicit dependencies on the transition and the reward models.

Since the optimal value function is also a value function for a policy, it must satisfy the Bellman equation. Here is the Bellman equation for V^* (Sutton and Barto 1998),

$$V^*(s) = \max_a Q^*(s, a) = \max_a (R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s')).$$

The Bellman equation for Q^* is,

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a'} Q^*(s', a')).$$

These formulations show the optimal value function, V^*, without a reference to any specific policy π. As a result, finding V^* is equivalent to solving a system of nonlinear equations, each of which is a Bellman equation with a state s; there are various methods for solving systems of nonlinear equations. For finite MDPs, it can be shown that such the system of Bellman equations have a unique solution, V^* (Sutton and Barto 1998). In appendix E, we describe a popular *value iteration* method for finding V^*.

2.1.3 Model-based reinforcement learning

An RL problem is formulated as an MDP. However, the transition model T and reward model R are unknown in an RL problem. Therefore, while the goal of an MDP solver is just an optimal policy, an RL method may have to concentrate on different things. In some domains, finding an optimal policy fast may be more critical than gaining a very

high cumulative reward, but in some other domains, this priority may be in reverse order. The goal of an RL method may also be a balance between these two interests.

There are two main approaches to solving an RL problem. The model-based approach explicitly learns the transition model T and reward model R and uses them to find an optimal policy via the Bellman equations (Equation 2.3). The model-free approach, on the other hand, updates state values based upon the temporal difference in the expected rewards between states, avoiding maintaining the two models. The following formula of value update (Rummery and Niranjan 1994) is an example,

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \alpha \left[R(s,a) + \gamma \max_{a' \in A} Q^{\pi}(s',a') - Q^{\pi}(s,a) \right],$$

where α is the agent learning rate, s' is the current state, and s is the previous state.

Advantages and disadvantages of these two approaches are difficult to judge as they depend on various assumptions and domain specific factors. However, we choose to study in depth the model-based approach since possessing the two models would likely open more chances to integrate expert knowledge and to generalize knowledge of the world quickly. Furthermore, knowledge between environments may be transferred conveniently and efficiently via the models. As a result, we would be able to achieve an autonomous agent that can learn a (near) optimal policy fast and gain high expected cumulative reward in a novel task.

Depending on the domain, steps in a model-based RL may be in different order, or may be implemented in numerous ways. Focusing on a method that may guide an agent both to gain high cumulative reward and to find a (near) optimal policy fast, we show a possible skeleton of a model-based RL algorithm in Algorithm 1. In the beginning, the transition model T and the reward model R can be initialized arbitrarily, or based on knowledge learnt in previous tasks. There are commonly four main steps in a model-based RL. The first step is to find an action policy π with the current available models T and R. This planning step may be done by a simple

Algorithm 1 Basic steps in a model-based RL algorithm

Input: S, A, T_0, R_0, γ

$T \leftarrow T_0$ //initialize T
$R \leftarrow R_0$ //initialize R
for $t = 0, 1, 2, ...$ **do**
  s_t denotes the current state
  $\pi \leftarrow$ Solve MDP using transition model T and reward model R
  $a_t \leftarrow ChooseAction(\pi, s_t)$
  Perform action a_t
  Observe next state s_{t+1} and reward r
  $T \leftarrow UpdateTransitionModel(s_t, a_t, s_{t+1})$
  $R \leftarrow UpdateRewardModel(s_t, a_t, r)$
end for

algorithm, *value iteration*. However, value iteration needs to update values of all the states in every iteration, which may be expensive and prohibitive in large state space applications. An agent may, instead, just partially back up values of a few previous states. Dyna (Sutton 1990) and Prioritized Sweeping (Moore and Atkeson 1993) are two partial-backup model-based RL algorithms proved successful in several applications.

The second step, that follows naturally after the planning step, is to perform an action. An action is usually chosen based on the learnt policy π, but sometimes it is chosen randomly based on some exploration strategy. Performing unplanned actions is needed because the current π may be sub-optimal; quality of the policy depends on T and R, but it is unknown whether the current models have yet approximated the true dynamics of the environment. One simple solution to this *exploration - exploitation* dilemma is an ϵ-greedy exploration strategy in which an agent will perform a random action with a small probability of ϵ at each state. With enough trials in the environment, the RL algorithm is guaranteed to converge to an optimal policy. Another popular technique is *Rmax* (Brafman and Tennenholtz 2002). The *Rmax* algorithm initializes every state and action Q-value with a maximum reward, and only updates Q-value of a state and action pair after the agent has "known" the pair. A pair is considered known if it has been tried frequently enough — more than a manually-set

parameter m. Consequently, the agent will be encouraged to explore every state and action to learn the models T and R. As a result, *Rmax* can guarantee a convergence time that is polynomial in the size of the state space (Brafman and Tennenholtz 2002). However, a drawback of this aggressive exploration strategy is that many losses or negative rewards may occur.

Once an action is performed, the agent will observe feedback from the environment, including reward and next state. Lastly, the feedback is exploited to update the transition and reward models; this fourth step is critical to model-based RL. These steps are repeated until a stopping criterion is met. For episodic tasks in finite MDPs, a common stopping criterion is when the agent reaches a terminal state.

2.2 Model representation

Model representation is key to an efficient model learning algorithm. It is more critical in the domain of model-based RL, where that learning has to be online, simple, and fast. We introduce here two approaches that are popular in the RL community, and focus our discussion on transition models. Reward models are usually simpler, but may also be represented in similar ways.

2.2.1 Tabular transition function

Recall that a transition model in MDP formulation is in the form of $T(s, a, s') = P(s'|s, a)$. It is, therefore, straightforward to organize a table to record observed transition events, and use maximum likelihood estimation technique (MLE) to learn the transition model. Figure 2-2, for example, presents two entries of the table. It shows that the agent has seen state s_3 in 45 times, and state s_2 in 15 times when doing an action a at state s_1. Probability of transiting to a state s' from a state s by an action a, via MLE, is,

$$T(s, a, s') = P(s'|s, a) = \frac{n(s, a, s')}{n(s, a)},$$

where $n(x)$ is the number of times the event x is observed. Figure 2-2 implies that $T(s_1, a, s_3) = 75\%$, and $T(s_1, a, s_2) = 25\%$.

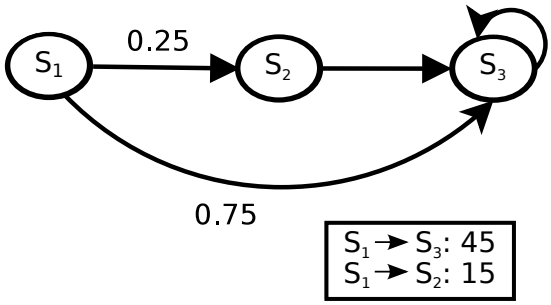


Figure 2-2: Two entries of a counting table representing a transition dynamics of an action a.

Depending on each application, a reward model can be represented in a similar or other ways. To estimate the expected reward $R(s, a) \mapsto \mathbb{R}$, we only need to maintain the average reward value for each event.

With the table representation, it is impossible for an agent to generalize knowledge across states. The agent has to frequently visit every state and try every action in the environment to learn a good transition or reward model. This disadvantage likely costs an agent much time to find a reasonable action policy. In practice, states usually share important characteristics contributing to the world transition dynamics.

2.2.2 Transition function as a dynamic Bayesian network

In order to share experiences across states, it is useful to identify a state by a vector of features, instead of an uninformative identifier. An environment modeled in this way is called a factored MDP. Let $\{S_1, S_2, ..., S_n\}$ be a set of discrete random variables, each S_i of which takes r_i different values. A state s in a state space S is then described by a $n-$dimensional feature vector $(S_1, S_2, ..., S_n)$. In factored MDP, it is impractical to model the world transition dynamics by a table, since the number of states is exponentially large in the number of features.

Kearns and Koller (Kearns and Koller 1999) suggested using dynamic Bayesian

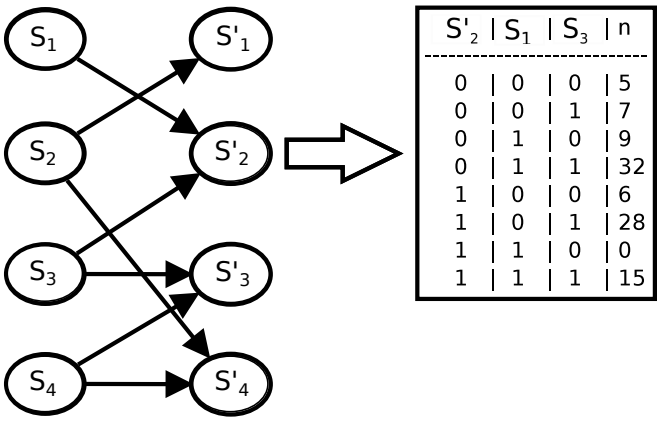


Figure 2-3: A DBN representing transition model by an action a.

networks (DBNs) to represent transition models. Figure 2-3 shows one example of DBN for factored state spaces. Each DBN represents a transition dynamics model for a single action. In a DBN, current and next states are represented by two separate layers. Each node is a state feature; for example S_1 denotes a feature of the current state, while S'_1 denotes the same feature at the next time step. There are two components forming a DBN, namely structure and parameters. Nodes are connected by arrows representing the dependence/independence relationships between features. Figure 2-3 implies that the probabilities of feature S'_1 can be estimated based on only the value of feature S_2 in the current state without caring about other features' values, i.e., $P(S'_1|S_1, S_2, S_3, S_4, a) = P(S'_1|S_2, a)$. To avoid computational complexities, most studies in RL domain assume DBNs to have no arrows between nodes within one layer - features at one time slice are conditionally independent given the previous state. Given the DBN structure, a transition probability can be efficiently factorized to conditional probabilities and calculated as below:

$$P(S'_1, ..., S'_n|S_1, ..., S_n) = \prod_{i=1}^{n} P(S'_i|Par(S'_i)),$$

where $Par(x)$ denotes parents of node x, or the nodes which have arrows going to node x. Action a is implicit in the above equation. Learning such conditional probabilities is easier than directly learning the joint probability $P(S'_1, ..., S'_n|S_1, ..., S_n)$ because

each conditional probability involves a much smaller number of features than the joint probability. For instance, the DBN in our sample suggests that:

$$P(S'_1, S'_2, S'_3, S'_4|S_1, S_2, S_3, S_4) = P(S'_1|S_2)P(S'_2|S_1, S_3)P(S'_3|S_3, S_4)P(S'_4|S_2, S_4).$$

Let r_i denote the number of different values of a feature S_i, and assume that the features have binary values, i.e., $r_i = 2$ for all S_i. Without DBN, to learn the joint probability, $P(S'_1, S'_2, S'_3, S'_4|S_1, S_2, S_3, S_4)$, we will need to consider $\left(\prod_{i=1}^{4} r_i - 1\right) \prod_{i=1}^{4} r_i = 15 \times 16 = 240$ parameters, while we only need to consider $(r_1 - 1) \times r_2 + (r_2 - 1) \times r_1 \times r_3 + (r_3 - 1) \times r_3 \times r_4 + (r_4 - 1) \times r_2 \times r_4 = 14$ parameters with the factorization in DBN.

The parameters component defines the conditional probabilities. They are typically represented by conditional tables which are local to each node in the second layer of the network. The right table in Figure 2-3 is an example encoding conditional probabilities of having certain values of S'_2 given values of feature S_1 and S_3; for instance $P(S'_2 = 0|S_1 = 0, S_3 = 0) = \frac{5}{5+6} = 0.45$, and $P(S'_2 = 1|S_1 = 0, S_3 = 0) = \frac{6}{5+6} = 0.55$. In some applications, it may be more efficient to represent those conditional probability tables or local structures by decision trees (Hester and Stone 2009).

Several studies including Kearns and Koller's study (1999) have demonstrated DBN-based RL algorithms to achieve better running times that can scale polynomially in the number of parameters of the DBN, which may be exponentially smaller than the number of states in a state space. However, most of the algorithms require that the DBN structures are readily available to an agent. DBN structure learning is generally intractable.

DBN has been a popular choice for factoring and approximating transition models. In DBN, structure learning or feature selection is equivalent to picking the parents of the state variables from the previous time slice. Recent studies have led to improvements in sample complexity for learning optimal policy. Those studies assume

maximum number of possible parents for a node (Strehl, Diuk, and Littman 2007; Diuk, Li, and Leffler 2009), or knowledge of a planning horizon that satisfies certain conditions (Chakraborty and Stone 2011). However, the improvements in sample complexity are achieved at the expense of actual computational complexity since these methods have to search through a large number of parent sets. Consequently, these methods appear feasible only in manually designed, low-dimensional state-spaces.

Instead of searching for an optimal model with a minimal number of samples at almost any cost, Degris et al. (2006) and Ross et al. (2008) attempt to save costs from early on, and gradually improve the model acknowledging that the true model may actually be unattainable. In this spirit, numerous practical applications could be considered, but unfortunately (Degris, Sigaud, and Wuillemin 2006) does not address online learning with large feature sets. Ross et al. (2008) use Markov chain Monte Carlo (MCMC) to sample from all possible DBN structures. However, the Markov Chain used has a very long burn-in period and slow mixing time, making sampling computationally prohibitive in large problems.

Kroon and Whiteson (2009) propose a feature selection method that works in conjunction with KWIK-Factored-Rmax to learn the structure and parameters of a factored MDP. This framework can extract a minimal set of features for representing the transition (and reward) model. Experiments show that planning on the reduced model yields improved performance. However, the computational cost of this method is still relatively high, since the KWIK-Factored-Rmax needs to search through a large combinatoric space of possible DBN structures to find the candidate structures for the proposed feature extraction method.

While there are often many states in the environments there are usually much fewer actions and their outcomes. Leffler et al. (2007) suggest to predict relative changes in states. In their relocatable action model (RAM), the idea is not to concentrate on estimating the state transition function $P(s' \mid s, a)$ directly, but on first estimating the outcome distribution of an action, $P(o \mid a, s)$ and then predicting the state transition

using a deterministic function η that maps the outcome and the pre-action state s to the next state $s' = \eta(s, o)$. The RAM gains efficiency by assuming that many states have similar outcome distributions for actions. In RAMs, the states are partitioned into classes so that all the states in a class share the similar outcome distributions. Therefore, we can write $P(o \mid a, s) = P(o \mid a, \kappa(s))$, where $\kappa(s)$ denotes the class of state s. The weakness of this method is that programmers have to manually input important features so that the agent can aggregate information from similar states for predicting action outcome.

Hester and Stone (2009; 2012) later extend this work (Leffler, Littman, and Edmunds 2007). They employ Quinlan's C4.5 (Quinlan 1993) to learn a decision tree for predicting relative changes of every state variable. This works better than the method by Degris et al. Despite adapting C4.5 for online learning, the method is still very slow as a costly tree induction procedure has to be repeated many times in a large feature space. In addition, all the data also needs to be stored for the purpose, which is undesirable in some applications. Besides, this method likely suffers from the problem of large state spaces because states are factored by a large set of features.

In Chapter 4, we will propose a customized DBN and a simple algorithm to learn the network structure and parameters incrementally and quickly. In Chapter 5, we will then introduce a model-based RL algorithm, which is based on our customized DBN, to work efficiently in case the environments contain very many irrelevant features.

2.3 Transfer learning

Having been long studied in psychological literature (Thorndike and Woodworth 1901; Skinner 1953), the insight behind transfer learning is that learning can be generalized across tasks. Numerous studies were later carried out to bring the idea to machine learning and reinforcement learning domain. Depending on characteristics of the tasks, knowledge transferred between tasks may be in various forms, such as a policy

(Maclin et al. 2005; Madden and Howley 2004), Q-values (Tanaka and Yamamura 2003), reward models (Wilson et al. 2007), or transition models (Atkeson, Moore, and Schaal 1997). Taylor and Stone present a relatively comprehensive survey of recent work (Taylor and Stone 2009).

An important problem in transfer learning is the negative transfer effect: transferred knowledge does not "help", but "harms" an agent. Learning for an optimal policy is decelerated with transferred knowledge. Instead of supporting an agent to gain high reward, transferred knowledge biases the agent to find low rewards. Since it is unknown how similar a novel task is to experienced tasks, the negative transfer effect is hardly avoidable. A good transfer learning method should not suffer much from negative transfer.

In RL, transfer learning is especially important because time for an RL agent to learn a (near) optimal policy in a task is usually very long. A transfer learning method in an RL domain is expected to reduce that learning time and improve performance of an RL agent in a novel task based on experience in previous tasks. The performance may refer to various aspects, such as the total reward the agent can achieve in its first actions, or the total reward accumulated over all its actions in the new task. An application may value one aspect higher than the others, thus defining a standard metric to evaluate a transfer learning method is difficult.

2.3.1 Measurement of a good transfer learning method

Currently no metrics have been accepted as standards for evaluating a transfer learning algorithm in RL, but some are commonly cited in recent studies:

- **Jumpstart:** the cumulative reward which an agent achieves at its first few steps in a task. In an episodic task, it is the total reward which an agent gains in the first episode. However, sometimes, jumpstart is also defined in a more relaxed way, referring to the reward in, instead of one, a few first episodes, as challenging environments likely prevent any method from performing well at

the very first try.

- **Asymptotic performance:** the final learned performance in terms of reward gain of an agent in a task. It measures the optimality of a policy which an algorithm can asymptotically find in infinite tries. In practice, one usually plots the cumulative rewards of a transfer and a non-transfer method episode by episode for many episodes. If both methods can converge to near (optimal) policies, one can then interpret how optimal each policy is and how fast each algorithm, with and without transferred knowledge, finds such policies. The metric, however, is not suitable for analyzing an agent's progress to a near (optimal) policy. The trade-offs of the agent's decisions in exploration and exploitation in early episodes could not be reflected. In case the methods cannot converge in permitted testing time, we can conclude very little about the methods via this metric.

- **Accumulated reward:** the total reward accumulated by an agent over every episode in a task. This metric is effective to measure an agent's performance in its whole life in a task. It captures the agent's gains and losses in every episode, and reflects the agent's progress over long run. Besides, when an algorithm converges, its cumulative reward gain in each episode will tend to be stable. Consequently, the curve showing the accumulated rewards over episodes also adopts a steady progression. Therefore, the metric is also useful for measuring convergence time of an algorithm to a policy as well as the optimality of that policy. This metric and the asymptotic performance metric are also commonly used in RL.

- **Transfer ratio:** the ratio of or the difference between the reward gains by two algorithms, e.g., a transfer and a non-transfer algorithms, or two different transfer learning methods. It estimates the relative performances of two methods.

- **Running time:** the total computer running time for an algorithm to finish certain requirements, such as finishing an episode, or a task. Since a transfer learning algorithm has the benefits of extra information from experienced tasks, it is expected to have better performance than non-transfer learning algorithm. However, in some situations, the higher reward achievement is not considered very significant if the algorithm takes much more time to process the transferred knowledge while in a task.

2.3.2 Review of existing transfer learning methods

An RL algorithm typically models a task by an MDP(S, A, T, R, γ). However, in a novel task most of RL algorithms start with all MDP components initialized randomly – random T, R, π, V or Q-function. Learning time of an RL algorithm is, therefore, usually very long. Hence, to improve performance of an RL algorithm in a task, a transfer learning method will concentrate on using knowledge experienced in previous tasks to initialize one or many of those components with values as close to the correct values as possible. Instead of initializing the values, it may also manage to bias the agent's decisions in some critical states to avoid losses or to achieve a near optimal policy faster.

We will next introduce recent methods showing how knowledge could be transferred between tasks in model-based RL. Model-free RL is out of the scope of this dissertation, so transfer learning methods specifically designed for model-free RL are not discussed.

In their study about transferring knowledge of transition models, Atkeson and Santamaria (Atkeson, Moore, and Schaal 1997) suggested to approximately represent a transition function by a locally weighted regression model (LWT) assuming that state and action spaces are factored and continuous. Focusing on the settings of homogeneous environments in which tasks share a common transition dynamics and a similar state representation, they successfully empirically demonstrated improvements

to jumpstart, accumulated reward, and asymptotic performance. Reward models were assumed known in their experiments. This study is among the very few that actually consider transferring the transition model to a new task (Taylor and Stone 2009). While their work is conducted in continuous state space using a fixed state similarity measure, it can be adapted to a discrete case. In Chapter 6, we argue that in heterogeneous settings the LWT strategy of trying to learn a single "perfect" model for transfer may have numerous problems.

Wilson et al. (Wilson et al. 2007) addressed the problem of transfer learning between environments where reward models may be different. The method is based on hierarchical Bayesian modeling to define a generative Dirichlet process over MDPs. For every new task, the reward model is then approximated by Gibbs sampling. This non-parametric framework allows an agent to group similar MDPs to a class and also to learn a new class for an MDP whose the reward model is different from the others. Thus, knowledge may be classified and used more efficiently in a novel task. Consequently, the method could successfully transfer knowledge of reward models in heterogeneous environments. However, MCMC method will usually have very slow mixing (Koller and Friedman 2009). The work by Wilson et al. demonstrates the method for reward models, and it is unclear how to extend the approach for transferring transition models. In Chapter 6, we propose another method that also groups similar MDPs to identify relevant experiences to leverage the learning process in a novel task. We show that our method can transfer transition models and it is computationally efficient.

Trying to bias an agent's exploration in a novel task, Fernández et al. (2010) transferred a library of policies learned in previous tasks. A working library to use in the novel task is the transferred library plus a new policy randomly initialized. In each trial episode, the agent will use a policy from the working library $\psi\%$ of the time to make actions, and follow ϵ-greedy exploration strategy $(1-\psi)\%$ of the time. ψ is manually set, and decreases over time. The new policy is updated at the same time

based on feedback from the environment by an RL method. After many episodes, that new policy will be selected if no old policies are "good" for this new task. In order to select a policy for use in an episode, they suggested to assign to each policy in the working library a probabilistic value estimating how good each policy is in the task. The probabilities are calculated based on the cumulative rewards which the agent gains in the trial episodes by using the policies, and on the number of times that each policy has been chosen. Experimental results showed that this method could achieve higher average accumulated rewards. However, this method requires the tasks to have the same state space, otherwise a state mapping strategy, which is usually difficult to establish, is needed. In Chapter 6, we will present an approach that also transfers knowledge in a form of a library. Unlike the policy library method, our method can transfer knowledge in heterogeneous environments without requiring a pre-defined state mapping function.

While superficially similar to our method, the case-based reasoning approaches (Celiberto et al. 2011; Sharma et al. 2007) focus on collecting good decisions instead of building models of world dynamics. Taylor et al. (2008) propose TIMBREL to transfer observations in a source to a target task. These methods, however, also need a manually tailored inter-task mapping to transfer in heterogeneous environments.

In brief, the number of studies about transfer learning in model-based RL is currently quite small. A rich body of them has just focused on automatically transferring between environments or tasks that have a similar state representation, a same state space, and share a common transition dynamics model. Otherwise, a manually tailored inter-task mapping is usually required. These assumptions, however, may not be satisfied in many real-life applications. For example, many environmental cues which help an agent navigate through forest are simply missing when the agent tries to navigate at sea. In addition, to the best of our knowledge, transferring transition models has not been studied comprehensively, although, as seen in MDP formulation and Algorithm 1, it has a key role in an efficient model-based RL.

2.4 Summary

We have briefly covered the background knowledge that is important for explaining our approaches to represent action learning and transfer learning in RL in the rest of this dissertation. We have also reflected on the limitations in the current studies in model-based RL. In the next chapter, we will outline a whole framework to overcome these limitations.

Chapter 3

An overview of the proposed framework

This chapter lays out the key steps in our unified learning framework to build an autonomous agent. The three subsequent chapters are methods to address these steps. We describe our approaches to learning and transferring transition models. Reward models are assumed known. The reward models, however, could be handled in analogous fashion.

Recent studies on using model-based RL to build autonomous agents in real-life applications face two important problems as summarized below:

1. The state of the art algorithms could not efficiently learn online the world dynamics models of complex and feature-rich environments.

2. No study seems to systematically discuss the problem of knowledge transfer in heterogeneous environments where dynamics models, feature distributions, state spaces, and terminal states may be different.

3.1 The proposed learning framework

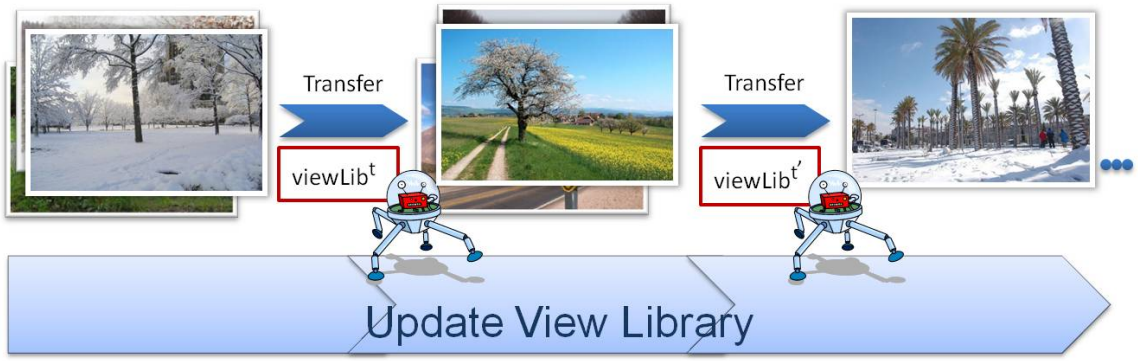


Figure 3-1: Our life-long learning agent.

The central idea in our approach is to maintain a library of *views* that the agent experiences in every task. A *view* consists of important features to model the transition dynamics of an action taken by the agent in an environment, and how those features affect the outcomes of the agent's actions. These views form the basis for the agent to quickly capture the transition dynamics in a new task; a view represents an expectation of the agent about the transition dynamics of its particular one (possibly several) action in the environment of a task. We then equip the agent with functions to select appropriate views to solve the task, and to learn new views if the environment is "new".

Figure 3-1 illustrates the life-long learning agent in our approach. We sketch the behavior of our agent in Algorithm 2. Initially when the agent does not have any information about the transition dynamics of the environment in a task, it selects views based on a historical record that tells how well each view in the library worked in previous tasks. The assumption is that the more frequently a view has worked in the past, the more possibly it will work again in a new task. The agent then operates according to the policy learnt based on the transition model built from those selected views. We assume that the reward model is known. However, since the views have been selected without any reference to the actual characteristics of this environment,

Algorithm 2 Overview of the proposed learning framework

Input: View library L.
Output: Updated view library L.

Initialize the system for a task
L^H : historical record of how good each view was in previous tasks
$L' \leftarrow L$ /*A WORKING COPY OF L TO FIND THE DYNAMICS OF THE CURRENT ENVIRONMENT*/
$B \leftarrow$ initialize belief of how well each view can approximate the current environment dynamics based on L^H.
/*————WHILE INTERACTING WITH THE ENVIRONMENT————*/
for $t = 0, 1, 2, ...$ **do**
  Select views to approximate the world transition dynamics
  $\{\mathbb{W}\} \leftarrow$ select the most promising views from L' based on B
  Interact with environment based on that approximate model
  $\pi \leftarrow$ plan an action policy based on $\{\mathbb{W}\}$
  s_t : current state in the environment
  $a_t \leftarrow$ choose an action to perform in s_t according to action policy π
  Perform action a_t and observe feedback: action outcomes from the environment
  Score all views with the new feedback
  $\mathbb{S} \leftarrow$ score all views in L' with the new feedback
  $B \leftarrow$ update belief about the views in L' based on the score $\mathbb{S}$
  Adjust all views with new feedback
  $L' \leftarrow$ Adapt all views toward this current environment based on the new feedback
  Break when the task ends
end for
Update view library L
$L \leftarrow$ Update the view library, e.g.
  If a view $\mathbb{W}$ is different from existing views in L,
    which means the transition dynamics of the corresponding action is new,
  then add that view $\mathbb{W}$ to L;
  else replace the old view in L with the newly updated view $\mathbb{W}$.

there would be high chances that those views are inappropriate in the current task. In other words, the transition model may be very insufficient to approximate the true transition dynamics in the current environment. As a result, the policy may be just bad, guiding the agent to lose rewards.

In order to limit such negative transfer effects, our agent exploits the outcomes or feedback for each of its actions, which is state changes (and rewards), to score all the views in the library. The score estimates the capability of each view in approximating the transition dynamics in the current environment, and it is a primary criterion to

re-select the views for subsequent decisions.

However, the task may also have an environment with very different transition dynamics, which none of the accumulated views are capable to capture. The environment feedback is also used to develop and incorporate new views into the library. These steps are repeated until a stopping criterion is met.

At the end of a task, the selected views will be recorded in the library if the transition dynamics which they are approximately representing is significantly different from existing ones in the library. Otherwise, the agent considers replacing the existing similar views. It also deletes long unused views because a large library will affect the agent's processing time.

To implement this framework, one has to face three main technical challenges. First, the transition model $T(S, A, S)$ is very task specific, which is probably a reason why there have not been many studies that transfers this model. Second, learning or updating a view or a transition model online in a complex and feature-rich environment is computationally expensive. Third, the view scoring method must be efficient and simple to calculate online based on feedback from the environment. Besides, updating the view library is also a non-trivial problem.

3.2 Summary

We have provided an overview of our learning framework, and characterized the major technical problems that we will need to tackle in order to build an autonomous lifelong learning agent. In Chapter 4, 5, and 6 respectively, we will discuss methods showing that it is possible to solve these problems. The whole system will be described again with technical details in Chapter 6.

Chapter 4

Situation calculus Markov decision process

This chapter introduces a variant formulation of MDP. An online multinomial logistic regression with group lasso to learn the transition model in this new formulation is also proposed. Content in this chapter has been published in Proceedings of the International Conference on Machine Learning (ICML'13) in 2013 (Nguyen et al. 2013).

In model-based RL, factored state representations, often in the form of DBNs, are deployed to exploit structure of the world dynamics. This allows the agent to plan and act in large state spaces without actually visiting every state. However, learning the world dynamics of a complex environment is very difficult and often computationally infeasible. Most recent work in this area is based on the RMAX framework (Brafman and Tennenholtz 2002), and focuses on sample-efficient learning of the optimal policies. This approach incurs heavy computational costs for maximizing information gain from every interaction, even in carefully designed, low-dimensional spaces.

We propose a variant formulation of the factored MDP that incorporates a principled way to compactly factorize the state space, while capturing comprehensive transition and reward dynamics information. We also propose an online multinomial logistic regression method with group lasso to automatically learn the relevant structure of the world dynamics model. While the regression models cannot capture the full conditional distributions like DBNs, their simplicity allows fast, online learning in very high dimensional spaces. Online feature selection is implemented with operating the regression algorithm in our variant MDP formulation.

In this chapter, we will first introduce an MDP representation that captures the world dynamics as action effects. We then present an online learning algorithm for identifying relevant features via sparse coding, which will learn a customized DBN of transition model in our new factored MDP.

4.1 Situation calculus MDP: CMDP

In a factored MDP, each state is represented by a vector of n state-attributes. The transition function for the factored states is commonly expressed using DBNs in which $T(s, a, s') = \prod_{i=1}^{n} P(s'_i | Pa_i^a(s), a)$, where Pa_i^a indicates a subset of state-attributes in s called the parents of s'_i (Fig.4-1a). Learning T requires learning the subsets Pa_i^a and the parameters for conditional distributions, or the DBN local structures.

Learning DBN structures of the transition function online, i.e., while the agent is interacting with the environment, however, is computationally prohibitive in most domains. On the other hand, recent studies (Xiao 2009; Yang et al. 2010) have shown encouraging results in learning the structure of logistic regression models, which can effectively serve as local structures in the DBNs even in high dimensional spaces. While these regression models cannot fully capture the conditional distributions, their expressive power can be improved by augmenting low dimensional state representation with non-linear features of the state vectors. We introduce an online

sparse multinomial logistic regression method that supports efficient learning of the structured representation of the transition function.

We present a variant of the factored MDP that defines a ”compact but comprehensive” factorization of the transition function and supports efficient learning of the relevant features. We consider two major aspects to modeling world dynamics: differentiating features and predicting changes.

First, we differentiate the roles of attributes or features that characterize a state. In a regular factored MDP, the state-attributes or features serve to both define the state space and capture information about the transition model. For example, two state-attributes, the (x, y)-coordinates uniquely identify a state and compactly factorize the state space in a grid-world. A policy can be learned on this factored space. The state transition dynamics, however, may depend on other features of the state, such as the surface material at the location (state). Such features are often carefully included in the state representations. While essential in formulating the transition or reward models, these features may complicate the planning or learning processes by increasing the size and complexity of the state space.

We separate the state identifying state-attributes from the “merely” informative state-features in our representation. This way, we can apply an efficient feature selection method on a large number of state features to capture the transition dynamics, while maintaining a compact state space.

Second, we predict the relative changes of states instead of directly specifying the next state-attributes in a transition. In the RL context, an action will stochastically create an effect that determines how the current state changes to the next one (Boutilier, Dearden, and Goldszmidt 2001; Leffler, Littman, and Edmunds 2007; Sherstov and Stone 2005). Mediating state changes via action effects is a common strategy in situation calculus (McCarthy 1963). Since the number of relative changes or action effects is usually much smaller than the size of the state space, or the size of state-attribute domains, the corresponding prediction task should be easier.

The learning problem can then be expressed as a multi-class classification task of predicting the action effects.

Formally, a situation calculus MDP (CMDP) is defined by a tuple $(S, f, A, T, E, R, \gamma)$, where S, A, T, R, γ have the same meaning as in a regular MDP. $S = \langle S_1, S_2, .., S_n \rangle$ is the state space implicitly represented by vectors of *n state-attributes*. The function $f : S \rightarrow \mathbb{R}^m$ extracts *m state-features* from each state. E is an *action effect* variable such that the transition function can be factored as

$$\begin{aligned} T(s, a, s') &= P(s' \mid s, a) = \prod_{i=1}^{n} P(s'_i \mid s, f(s), a) \\ &= \prod_{i=1}^{n} \sum_{e \in E} P(s'_i \mid e, s) P(e \mid s, f(s), a). \end{aligned}$$

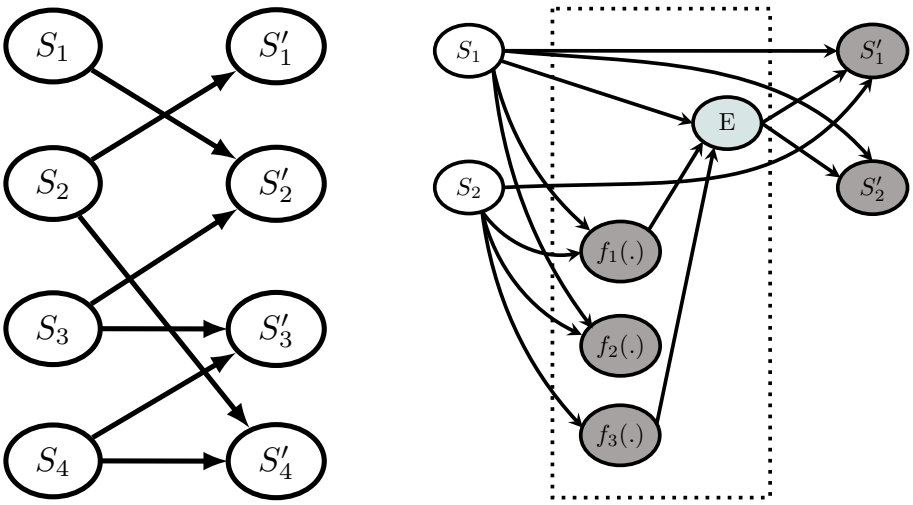


Figure 4-1: a.) Standard DBN. b.) Our customized DBN for CMDP.

Figure 4-1b shows an example of this decomposition. The agent uses the feature function f to identify the relevant features, and then uses both state attributes and features to predict the action effects. We also assume that the effect e and current state s determine the next state s', thus $P(s'|e, s)$ is either 0 or 1. This defines the semantic meaning of the effect which is assumed to be known by the agent. The remaining task is to learn the $P(e|s, a) = P(e|x(s), a)$, where $x(s) = (s, f(s))$, which is a classification

problem; we propose to solve this problem by using multinomial logistic regression methods.

4.2 mDAGL: multinomial logistic regression with group lasso

We will next introduce a regularized online multinomial regression method with group lasso that allows us to learn a probabilistic multi-class classifier with online feature selection. We also show that the structure and parameters of the learnt classifier are likely to converge to those of the optimal classifier.

4.2.1 Multinomial logistic regression

Multinomial logistic regression is a simple yet effective classification method. Assuming K classes of d-dimensional vectors $x \in \mathbb{R}^d$, we represent each class k with a d-dimensional prototype vector W_k. Classification of an input vector x is based on how "similar" it is to the prototype vectors. Similarity is measured with inner product $\langle W_k, x \rangle = \sum_{i=1}^{d} W_{ki} x_i$, where x_i denotes feature i. The log probability of a class is defined by $\log P(y = k|x; W_k) \propto \langle W_k, x \rangle$. The parameter vectors of the model form the rows of a matrix $W = (W_1, ..., W_K)^T$.

Let $l_t(W^t) = -\log P(y^t|x^t; W^t)$ denote the item-wise log-loss of a model with coefficient matrix W^t predicting a data point (y^t, x^t) observed at time t. A typical objective of an online learning system is to minimize the total loss by updating its W^t over time. However, the resulting model will often be very complicated and over-fitting. To achieve a parsimonious model, we express our a priori belief that most features are irrelevant or superfluous by introducing a regularization term $\Psi(W) = \lambda \sum_{i=1}^{d} \sqrt{K} \|W_{\cdot i}\|_2$, where $\|W_{\cdot i}\|_2$ denotes the 2-norm of the i^{th} column of W, and λ is a positive constant. This regularization is similar to that of group lasso. It communicates

the idea that it is likely that a whole column of W has zero values (especially, for large λ). A column of all zeros suggests that the corresponding feature is not necessary for classification.

The objective function can now be written as

$$\begin{aligned} L(T) &= \sum_{t=1}^{T} l_t(W^t) + \Psi(W^t) \\ &= \sum_{t=1}^{T} -\log \frac{e^{\left\langle W^t_{y^t}, x^t \right\rangle}}{\sum_k e^{\left\langle W^t_k, x^t \right\rangle}} + \lambda \sum_{i}^{d} \sqrt{K} \|W^t_{\cdot i}\|_2, \end{aligned}$$

where W^t is the coefficient matrix learned using $t-1$ previously observed data items. The quality of a sequence of parameter matrices $W^t, t \in (1, \ldots, T)$ with respect to a fixed parameter matrix W can be measured by the amount of extra loss, or regret

$$\begin{aligned} R_T(W) &= L(T) - L_W(T) \\ &= \sum_{t=1}^{T} (l_t(W^t) + \Psi(W^t)) - \sum_{t=1}^{T} (l_t(W) + \Psi(W)). \end{aligned}$$

We want to learn a series of parameters W^t to achieve small regret with respect to a good model W that has a small loss $L_W(T)$.

4.2.2 Online learning for regularized multinomial logistic regression

We introduce *mDAGL* (Algorithm 3) to extend the efficient dual averaging method (Xiao 2009) for solving lasso and group lasso (Yang et al. 2010) logistic regression on binary classification to the multi-class case.

Let $h(W)$ be a strongly convex function with modulus 1; $W^0 = \arg\min_W h(W)$, and let $W^{t=1}$ be initialized to W^0. Let G^t_{ki} be the partial derivatives of function $l_t(W)$ with respect to W_{ki} at W^t ($G^t_{ki} = \frac{\partial l_t}{\partial W_{ki}}(W^t)$). We define $\bar{G}^t$ to be a matrix of average partial

derivatives, i.e., $\bar{G}^t_{ki} = \frac{1}{t}\sum_{\tau=1}^{t} G^\tau_{ki}$, where

$$G^\tau_{ki} = -x^\tau_i(I(y^\tau = k) - P(k|x^\tau; W^\tau)). \tag{4.1}$$

For any data observed at time t, we update the coefficient matrix via

$$W^{t+1} = \arg\min_W \left(\left\langle \bar{G}^t, W \right\rangle + \Psi(W) + \frac{\beta_t}{t} h(W)\right), \tag{4.2}$$

where β_t is a non-negative, non-decreasing constant sequence, and $\langle \cdot, \cdot \rangle$ denotes an inner product between two matrices; $\left\langle \bar{G}^t, W \right\rangle = \sum_{k,i} \bar{G}^t_{ki} W_{ki}$.

Theorem 1 (Update Rule)**.** *Given $h(W) = \frac{1}{2}\|W\|_2^2$, a $K \times d$ average gradient matrix $\bar{G}^t$, and a regularization parameter $\lambda > 0$, the optimal solution of (4.2) is achieved column-wise as follows*

$$W^{t+1}_{\cdot i} = \begin{cases} \overrightarrow{0} & \text{if} \quad \|\bar{G}^t_{\cdot i}\|_2 \leq \lambda\sqrt{K}, \\ \frac{t}{\beta_t}\left(\frac{\lambda\sqrt{K}}{\|\bar{G}^t_{\cdot i}\|_2} - 1\right)\bar{G}^t_{\cdot i} & \text{otherwise.} \end{cases} \tag{4.3}$$

Proof Sketch. Since the minimization problem 4.2 is component-wise on one column of W, we can focus on each of the column of W separately to find its solution. Because inner product of two vectors of same length will have smallest value when the two vectors are in opposite direction, the solution to each of the component-wise minimization problem should be a factor of φ ($\varphi \leq 0$) to the corresponding column in the average gradient matrix. Subsequently, we can turn the problem into a basic quadratic function minimization problem. □

This rule dictates that when the length of the average gradient matrix column is small enough, the corresponding parameter column should be truncated to zero. This amounts to feature selection.

The following regret analysis confirms that the solution will converge and that the average maximal regret asymptotically approaches zero with rate $O(\frac{1}{\sqrt{t}})$.

Theorem 2 (Regret Bound)**.** *Let the sequence of* $\{W^t\}_{t\geq 1}$ *be generated by the update rule (4.3), and assume that there exists a constant G such that* $\|G^t\|_2^2 \leq G^2, \forall t \geq 1$. *If we choose* $\beta_t = \alpha\sqrt{t}$ *where* $\alpha > 0$*, then for any* $t \geq 1$ *and for any W that satisfies* $h(W) \leq D^2$ *where D is a constant, the average regret is bounded as*

$$\frac{R_t(W)}{t} \leq \frac{\Delta}{\sqrt{t}}, \quad t = 1, 2, 3.., \tag{4.4}$$

where $\Delta = \left(\alpha D^2 + \frac{G^2}{\alpha}\right)$.

Proof Sketch. The item-wise loss function $l_t(W)$ of multinomial logistic regression is convex, thus the techniques used for binary case (Xiao 2009) can be applied for multinomial case as well. □

Algorithm 3 The *mDAGL* algorithm

Input: λ, α

Let $h(W)$ be a strongly convex function with modulus 1
Let $W^1 = W^0 = \arg\min_W h(W)$
Let $\bar{G}^0 = \bar{0}$
for $t = 1, 2, 3, ...$ **do**
  $(y^t, x^t) \leftarrow$ observe data
  $(W^{t+1}, \bar{G}^t) \leftarrow$ mDAGL-update$(t, y^t, x^t, W^t, \bar{G}^{t-1}, \lambda, \alpha)$
end for

Algorithm 4 mDAGL-update

Input: $t, y^t, x^t, W^t, \bar{G}^{t-1}, \lambda, \alpha$

$G^t \leftarrow$ use equation 4.1 with $(y^t, x^t), W^t$
$\bar{G}^t \leftarrow \frac{t-1}{t}\bar{G}^{t-1} + \frac{1}{t}G^t$
$W^{t+1} \leftarrow$ use equation 4.3 with $\bar{G}^t, \beta_t = \alpha\sqrt{t}, \lambda$
return $(W^{t+1}, \bar{G}^t)$

Since the average regret goes asymptotically to zero, it may look very feasible that the sequence (W^t) also converges to some optimal W^*. However, the regret analysis is valid for any sequence of data, and without additional assumptions about the data generating process there may not be any asymptotically optimal classifier W^*, thus

convergence is not meaningful. To study convergence, we assume the data is to be sampled independently from some joint distribution p for data vector (y,x). In this case we try to find a W that minimizes the expected loss $E_p[l(W)] + \Psi(W)$. Now assuming that the optimal solution W^* is sparse, and some other technical assumptions, it is indeed possible to show that

$$P(\|W^t - W^*\|_2 > \epsilon) < \left[\epsilon^{-1}(\epsilon^{-1} + r^{-1}) + \frac{2}{c}\Delta\right] t^{-\frac{1}{4}}, \tag{4.5}$$

where r and c are constants (see Lemma 13 in (Lee and Wright 2012) for the result and its assumptions).

4.3 An example

We need to implement an autonomous robot to navigate in a small garden that is colorful with the red of tomatos, the green of vegtables, and the black of soil, etc. The garden surface is made of various materials such as sand, soil, or rock. The robot has two actions, namely ”move left”, and ”move forward”. Depending on the characteristics at each location in the garden, an action may have different effects on the environment. For instance, the action move left may take the robot to a new location on its left, or on its front depending on the surface at the robot place.

In order to learn an action policy for this task, one may simply discretize the space into a grid and learn a policy for the robot to navigate in that grid world. Two variables, the x and y coordinates of the robot in the garden, can fully identify every state in the environment, thus CMDP factorizes the state space by these two variables, which we call *state-attributes*. The transition model is, then, a function to encode the probability distributions of these two variables at a next time step given their current values. Other information such as the colors of the trees, tomatos, surface materials etc. that may help to learn the world transition dynamics is represented as *state-features*.

In some applications, one can represent the transition dynamics models efficiently

via our customized DBN (in Chapter 4) based on only these state-attributes and state-features; a state attribute is predicted directly based on information in the previous time slice. There are no nodes of action effects in the network. However, in some other domains, we may further exploit domain knowledge of robot actions to model the transition dyanmics more compactly. In this example, An attempt to move left in the grid world may lead the agent to move one step left or one step forward, with small probabilities. The relative changes in states, "moved left" and "moved forward", are called *effects* of the action. Thus, instead of directly predicting state-attributes, we can first predict the action effects, and then infer next state-attributes based on current values of state-attributes and the effects.

Assume that each cell in this environment has one of five surface materials: sand, soil, water, brick, and grass; there may be walls between cells. Surface and walls are features that determine the stochastic dynamics of the world. In CMDP, for the action "move up" we then may determine the probabilities of different effects using a W matrix below, for instance. The columns of the matrix correspond to the 9 indicator variables and a bias factor (brick, sand, soil, water, grass, wall-up, wall-left, wall-bottom, wall-right, bias) and rows correspond to possible effects for movements (up, left, down, right, not moved).

$$W = \begin{pmatrix} 3.99 & 3 & 3.5 & 2.6 & 0 & -4 & 0 & 0 & 0.01 & 0 \\ 1.23 & 1.1 & 1.15 & 1.2 & 0 & 0 & -4 & 0.02 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.03 \\ 1.23 & 1.1 & 1.15 & 1.2 & 0.01 & 0 & 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0.9 & 0.01 & 0.91 & 0 \end{pmatrix}$$

The probability that the action "move up" leads to moving left instead depends on the feature vector x of the cell. Feature-vector x (a column vector) describing the cell determines this probability via $P(left; W, x) = exp(Wx[2])/Z$, where Z is the normalizing constant ensuring that probabilities sum to one and the [2] picks the

second entry of the vector Wx (second, since "left" is the second effect in order).

For example, for a sand cell with walls up and right, vector of relevant features $x = (0, 1, 0, 0, 0, 1, 0, 0, 1, 1)$. We have:

$$\begin{aligned}
&Wx[1] = 3 - 4 + 0.01 = -0.99; && \exp(-0.99) = 0.37 \\
&Wx[2] = 1.1; && \exp(1.1) = 3.00 \\
&Wx[3] = 0.03; && \exp(0.03) = 1.03 \\
&Wx[4] = 1.1 - 4 = -2.9; && \exp(-2.9) = 0.06 \\
&Wx[5] = 4 - 0.91 = 3.01; && \exp(3.01) = 20.29
\end{aligned}$$

Normalizing the the numbers $\exp(Wx[i])$ gives us the effect distribution,

$$P(E; W, x) = (0.015, 0.121, 0.042, 0.002, 0.820).$$

We want to further disallow any chance of moving through the walls so we distribute the probability of these effects (0.015 for up and 0.002 for right) evenly to other ones. After this the final effect distribution is,

$$\begin{aligned}
P(E; W, x) &= (0, 0.13, 0.05, 0, 0.83); \\
P(left; W, x) &= 0.13.
\end{aligned}$$

In practice, there may be a few important features modeling the transition dynamics. However, an autonomous agent do not know those features. It has to first consider many features, which may be directly observed in the environment through its sensors, and be generated from those features, before learning to use only the important ones. Consequently, the W matrix may be very large and computationally expensive to learn. For instance, in our example, in addition to 9 indicator features + bias, the robot has to prepare a vector x including many other irrelevant features that are not shown

here, such as the colors, the fruits. However, by incorporating sparse coding into this customized DBN, we will be able to efficiently learn to select features to have a compact W in which only columns corresponding to important features have values different from zero; columns in the weight matrix W that correspond to irrelevant features are all zeros.

4.4 Summary

The core idea in our variant formulation of MDP, called CMDP, is to identify a state space by state-attributes, while using state-features as extra information to predict the transition dynamics between state-attribute values given an action via multinomial logistic regression models. As shown, a multinomial logistic regression model and its structure can be learnt efficiently online with the *mDAGL* algorithm as proposed. As a result, one can conveniently factorize a state space with a large number of features and let the agent to learn the transition dynamics model online (including both paramters and structure).

CMDP also suggests a way to formulate the state transition dynamics model via action effects. In some applications, this decomposition strategy may be quite useful, since the number of possible action effects are often smaller than the number of states or state-attributes.

In the next chapter, we will introduce a new RL algorithm based on this new MDP formulation. The advantages and disadvantages of CMDP will also be empirically evaluated and discussed.

Chapter 5

Model-based RL with online feature selection

We next present *loreRL*, a new model-based RL algorithm, in which a task is modeled by a CMDP. We demonstrate, both by theory and empirical experiments, that our method leads to computationally efficient learning of a near optimal policy in complex and feature-rich environments. All these results have been published in Proceedings of the International Conference on Machine Learning (ICML'13) in 2013 (Nguyen et al. 2013).

5.1 loreRL: the model-based RL with multinomial logistic regression

Our main task is to turn transition model learning into the learning of conditional distributions $P(E \mid s, f(s), a)$ using multinomial logistic regression for which attention to relevant features can be efficiently implemented online via *mDAGL*.

Algorithm 5 The *loreRL* algorithm

Input: mDAGL regularization parameters λ, α;
CMDP variables S, f, A, E, R, γ; exploration ϵ

Let $h(W)$ be a strongly convex function with modulus 1
Let $W^0 = \arg\min_W h(W)$
Let $\bar{W} = (W_1, W_2, \ldots, W_{|A|}) = (W^0, W^0, \ldots, W^0)$
Let $\bar{G} = (\bar{G}_1, \bar{G}_2, \ldots, \bar{G}_{|A|}) = (\bar{0}, \bar{0}, \ldots, \bar{0})$
$s_0 \leftarrow$ random initial state
for $t = 1, 2, 3, \ldots$ **do**
  $\pi \leftarrow$ Solve MDP using transition model $T(\bar{W})$
  $a \leftarrow \pi(s^t, \epsilon)$ #ϵ-greedy action selection
  Take action a yielding effect e, next state s^{t+1}
  $(W_a, \bar{G}_a) \leftarrow mDAGL - update(e, s^t, f(s^t), W_a, \bar{G}_a, \lambda, \alpha)$
end for

The key steps of our method, called *loreRL* (RL with regularized logistic regression), are presented in Algorithm 5. Inputs to *loreRL* are the CMDP components (except the transition function), regularization parameters λ and α of the *mDAGL* algorithm, and the ϵ that determines the probability of taking a random action. We first initialize logistic regression parameters W_a and the average gradient matrices $\bar{G}_a$ for each action $a \in A$. We also randomly select a starting state s^0.

At each time step, a random action a is chosen with a small probability ϵ, but otherwise we calculate the optimal policy π for an MDP with the transition model $T(W)$ based on the current effect predictors. While in this dissertation we have used value iteration (like in *Rmax*) for finding an optimal policy, any other planning technique can be used as well. We do not focus on the planning part of RL here, but Dyna-Q (Sutton 1990), Prioritized Sweeping (Moore and Atkeson 1993), or UCT (Kocsis and Szepesvári 2006) can be deployed for a more scalable algorithm. After performing an action a in state s^t and observing its effect e, the experience $(e, s^t, f(s^t))$ will be presented to the *mDAGL* algorithm that updates the gradient matrix $\bar{G}_a$ and the parameter matrix W_a.

As we just do ϵ-greedy random sampling, it is impossible to guarantee PAC convergence to an optimal policy. Assuming that observed data is i.i.d, we can prove that

difference in optimal value functions of two CMDPs with different logistic regression based transition functions is bounded by the difference in their parameters.

Theorem 3 (Difference in Value Function)**.** [1] *Let* $M_1 = (S, f, A, T(W^{M_1}), E, R, \gamma)$ *and* $M_2 = (S, f, A, T(W^{M_2}), E, R, \gamma)$ *be two CMDPs with optimal policies* π_1 *and* π_2 *respectively. Let us denote by* V_π^M *the value function for policy* π *in CMDP* M*. Let*

$$\epsilon_1 = 2\sqrt{\max_{a\in A, e\in E} \|W_e^{(a),M_1} - W_e^{(a),M_2}\|_1 \sup_s \|x(s)\|_1},$$

then $\max_{s\in S}\left(V_{\pi_1}^{M_2}(s) - V_{\pi_2}^{M_2}(s)\right) \leq \frac{2\gamma V_{\max}\epsilon_1}{1-\gamma}$,

where $W_e^{(a),M_1}$ *and* $W_e^{(a),M_2}$ *refer to the vector of coefficients corresponding to class* $E = e$ *under action* a *in model* M_1 *and* M_2 *respectively,* $\|\cdot\|_1$ *is the 1-norm of vector, and* V_{max} *is the maximum value of any state for any policy in either of the CMDPs.*

Proof Sketch. By Pinsker inequality, we can bound the probability prediction difference between the transition models in two CMDPs M_1 and M_2 by ϵ_1. The bound on the difference between the value functions is then derived based on Lemma 33 in (Li 2009). □

By taking M_2 to be an CMDP based on the optimal W^* and M_1 an estimated CMDP based on *mDAGL*, the vanishing bound given in equation (4.5) can be translated into a vanishing bound for value difference of policies. This leads to a corollary for convergence to optimal policy.

Corollary 1 (Convergence guarantee)**.** *If the transition model in the true CMDP* M_2 *is representable by multinomial logistic distributions, then the policy* π_1 *calculated based on CMDP* M_1*, where the transition model is learnt by* mDAGL*, would most probably optimal.*

When we cannot express the true transition dynamics as logistic regression based on the available state features, it is hard to give guarantees of performance. We

[1] Full proof of this theorem, attached in Appendix C, has been written by Zhuoru Li.

will have a case study on a real robotic domain, where we cannot expect the world dynamics to follow logistic distributions, and the data observation to be i.i.d., to backup our theories (see Section 7.4.1). However, we can still have some confidence in doing well here. The logistic regression model P_l^* closest (in Kullback-Leibler distance) to the true model P_{true} (possibly not a logistic regression model) is the one[2] that has the smallest expected log-loss. While our optimality criterion is the expected regularized log-loss, we expect the regularized log-loss optimal model P_Ψ^* to be close to P_l^* thus almost as close to P_{true} as we can get. This relatively small KL-distance can be converted to relatively small distances in actual transition probabilities, which can then further be converted to a relatively small bound on value differences by the same arguments used in proving Theorem 3. Therefore, since our model would very likely converge close to P_Ψ^*, we can expect to do almost as well as P_Ψ^*.

5.2 Experiments

We present the empirical evaluation of *loreRL* on a popular domain of grid-world navigation. The experiments aim to demonstrate that *loreRL* can a) generalize and approximate the transition model to achieve fast convergence to near optimal policy, and b) with feature selection, perform well in complex, feature rich environments. We also want to see if theoretical promises derived under assumption of i.i.d sampling can be realized in practice. We compare accumulated rewards of *loreRL* with factored Rmax (*fRmax*), in which the network structures of transition models are known (Strehl, Diuk, and Littman 2007), and with factored ϵ-greedy (*fEpsG*), in which the optimistic Rmax exploration of *fRmax* is replaced by ϵ-greedy strategy. We also compare our method with RL-DT (Hester and Stone 2009) and LSE-Rmax (Chakraborty and Stone 2011), which are the state of the art model-based RL algorithms for learning transition models. All the results are averaged over 20 runs, and we report the 95% confidence

[2]Such model may not always exist since the parameter set is open. However, for our argument, any model with almost infimum distance to the true model will do.

intervals. We run the experiments with *loreRL* having $\alpha = 1.5$, $\lambda = 0.05$, $\gamma = 0.95$, exploration $\epsilon = 0.05$, parameter $m = 10$ for *fRmax*, $m = 5$ for *Rmax* ($m = 5$ is small for *Rmax*, but increasing it has not yielded better result), fixed $m = 10, \sigma = 0.99$ for *LSE-Rmax* (values originally used in (Chakraborty and Stone 2011)).

5.2.1 Experiment set-up

In this domain, the agent tries to reach its goal in the grid-world consuming as little energy as possible. The world has 900 cells/states. Each cell has one of five surface materials: sand, soil, water, brick, and fire; there may be walls between cells. Surface and walls are features that determine the stochastic dynamics of the world. In addition, to test the variable selection aspect, we attach hundreds of random binary features to the environment ; the agent needs to focus on the important features to learn the environment dynamics model, and consequently to achieve its goal. The agent can perform four actions (move up, down, left, right), which will lead it to one of the four states around it or leave it to its current state. Effects of the actions are captured in five outcomes (moved up, left, down, right, did not move). The states are defined by the (x,y)-coordinates of the agent. To perform an action, agent will spend 0.01 units of energy. It loses 1 unit if falling into a state of fire, but gains 1 unit when successfully reaching an exit door. A task ends when agent reaches a terminal state, i.e., any exit door or state with fire.

We generate the environment transition models from four random multinomial logistic distributions (one for each action) (see Appendix D); every different combination of cell surfaces and walls around the cell will lead to different transition dynamics at the cell. The probability of going through a wall is rounded to zero and the freed probability mass is evenly distributed to other effects. The example in Chapter 4 provides a detailed description. The agent's starting position is randomly picked in each episode.

5.2.2 Generalization and convergence

We first show that when the feature space is small, *loreRL* performs as efficiently as the state of the art methods. *RL-DT* employs a decision tree to generalize transition dynamics knowledge over states, but it is implemented with ϵ-greedy exploration strategy. *LSE-Rmax* appears to be the best structure learning method for ergodic factored MDPs (Chakraborty and Stone 2011). *fRmax* and *fEpsG* have correct DBN structures provided by an oracle. All the methods are implemented with our customized DBN to utilize domain knowledge. *Rmax* is included as a reference to show the effect of knowledge generalization.

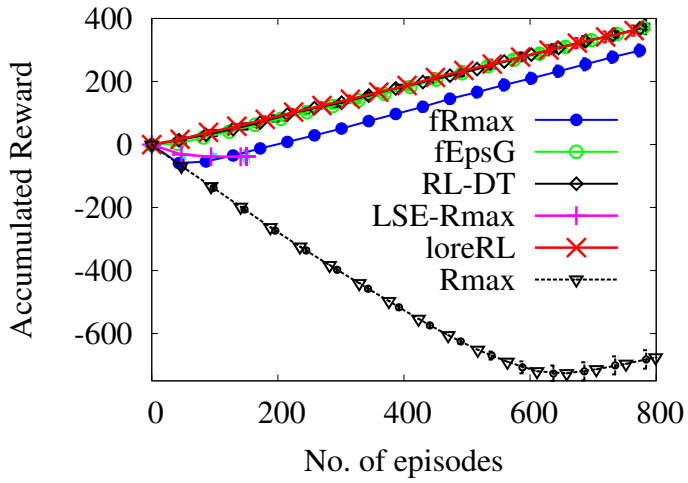


Figure 5-1: Accumulated reward in a CMDP with 10 features.

As seen in Figure 5-1, *loreRL* can approximate the world dynamics using samples in all the states, thus it converges as fast as *fEpsG*, and *RL-DT* to near optimal policy. Although *fRmax* is provided with the correct DBN structure, its accumulated reward is lower due to aggressive exploration to find the optimal model. After exploration the policy is guaranteed to be near optimal, but it may still take a long time (or forever) to catch up with *loreRL*. While *LSE-Rmax* follows the *Rmax* scheme, it starts with a simple model and explores a bit less aggressively than *fRmax*, gaining some advantage in early episodes. However, *LSE-Rmax* appears to require much more data to choose a

more complex model. Its accumulated reward drops below *fRmax* after 150 episodes, and the angle of the curve suggests that its DBN structure is still not correct. We do not run *LSE-Rmax* for more episodes, as the algorithm has very high computational demand (Table 5.1).

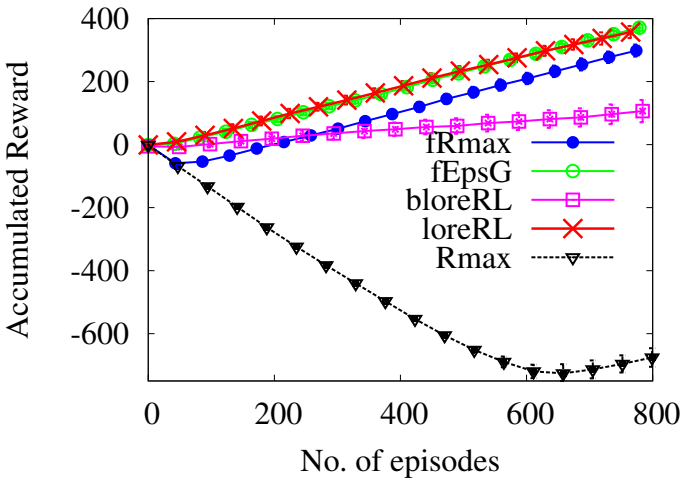


Figure 5-2: Accumulated reward in a CMDP including extra 200 irrelevant features.

When the feature set has many irrelevant features (Figure 5-2), *loreRL* is able to learn the relevant ones and still gain nearly as high accumulated reward as *fEpsG* which has relevant features provided by oracle. Also *loreRL*'s running time is not much longer than *fRmax*'s or *fEpsG*'s (Table 5.1). Other methods are too slow to be run in this high-dimensional environment.

These results also suggest that with ϵ-greedy exploration and random restarts, near optimal policy can be found even without i.i.d data sampling.

Table 5.1: Average running time per. episode in 800 episodes when acting in an environment with 210 features. (Slow *RL-DT, LSE-Rmax* could only be run with 10 features.) Run on Intel Xeon CPU 2.13GHz, 32GB RAM.

Algorithm	*fRmax*	*fEpsG*	*RL-DT*	*LSE-R.*	*bloreRL*	***loreRL***
Time (sec.)	0.26	0.25	9.09	67.53	4.3	0.55

5.2.3 Feature selection

To model real-life situations, the feature space is usually exponentially large, as we have discussed in section 1.2. The ability to focus only on the (most) relevant features is required to achieve effective learning. To understand the role of attention, we focus on comparing *loreRL* with a *bloreRL* that is based on multinomial logistic regression without feature selection. *fEpsG* and *fRmax* are base lines.

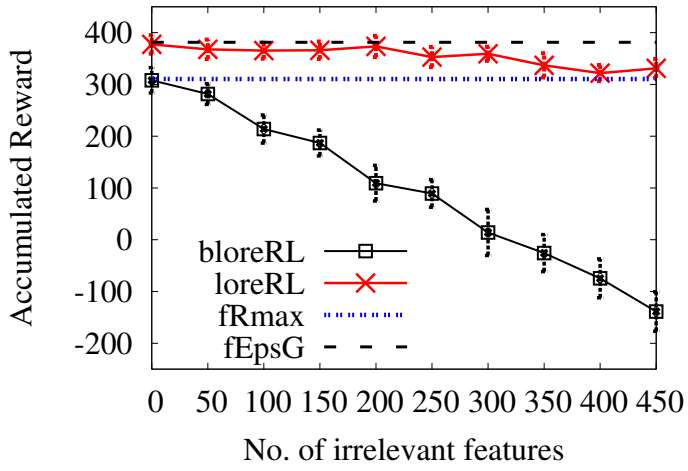


Figure 5-3: Accumulated rewards achieved after 800 episodes in CMDPs with different no. of irrelevant features. The CMDPs formulate the same grid-world, but use different sets of features.

Figure 5-2 shows the accumulated reward when the environment has 200 irrelevant binary features. As seen, *loreRL* is still able to converge fast to optimal policy, and outperforms *fRmax* and *bloreRL*. Figure 5-3 shows performances of *loreRL* and *bloreRL* after 800 episodes as a function of the number of irrelevant features. Only minimally affected by the actual number of irrelevant features, *loreRL* can quickly select the relevant features and outperform *bloreRL*. *loreRL* does not lose much to *fEpsG* either. While *fRmax* may find an optimal policy before *loreRL* due to agressive exploration, its accumulated reward is still lower than *loreRL*'s. In our experiments, we also observed that *loreRL* is much faster than *bloreRL* (see Table 5.1). That is probably due to *loreRL*' capability of selecting a small set of features. .

5.3 Discussion

We have demonstrated how online multinomial logistic regression with group lasso can be used to quickly obtain a parsimonious transition model in model based RL. The method leads to fast learning since a single transition model can be learnt using samples from all the states with a small set of features. Strehl and Littman (2007), and Walsh et al. (2009) have also proposed an online linear regression method with L2-regularization to approximate the transition model in continuous MDP. L2, however, does not implement feature selection.

The efficiency is gained, however, at the expense of generality. Not all transition functions can be accurately represented as predicting action effects using state features via logistic regression. Nevertheless, we believe that this compromise between scalability and generality is often a useful one. The generality problem may also be alleviated by introducing non-linear features that are combinations of the original ones.

The current work addresses the question of learning good models, but the problem of learning good policies in large state spaces still remains. Our model learning method is independent of the policy learning; thus, it can well be coupled with any scalable approximate policy learning algorithms.

5.4 Summary

In this chapter, we have introduced *loreRL*, a new model-based RL algorithm, to work with complex and feature-rich environments. In *loreRL*, a task is modeled by a CMDP, and the transition dynamics model is learnt by *mDAGL*. Theoretical and empirical evidence suggests that CMDP and *mDAGL* are useful components to implement "attention focus" or online feature selection for model-based RL. As we will see in the next chapter, a *view* in our complete framework (in Chapter 3) can be automatically learnt by *loreRL*, or concretely through CMDP and *mDAGL*.

Chapter 6

Transferring expectations in model-based RL

We now introduce a technique enabling an agent to accumulate experience during its life time to quickly adapt to a new task. To address the challenging settings of heterogeneous environments, we propose a transfer learning framework based on transferring expectations using a library of views. Each view is constructed upon CMDP and *loreRL* method as previously described in Chapters 4 and 5. This approach has been published in the International Conference on Machine Learning Workshop on Representation Learning (ICML'12) in 2012 (Nguyen, Silander, and Leong 2012a), and in Proceedings of Advances in Neural Information Processing Systems (NIPS'12) in 2012 (Nguyen, Silander, and Leong 2012b).

In RL, an agent autonomously learns how to make optimal sequential decisions by interacting with the world. The agent's learned knowledge, however, is task and environment specific. A small change in the task or the environment may render the agent's accumulated knowledge useless; costly re-learning from scratch is often needed.

Transfer learning addresses this shortcoming by accumulating knowledge in forms that can be reused in new situations. Many existing techniques assume the same state space or state representation in different tasks. While recent efforts have addressed inter-task transfer in different action or state spaces, specific mapping criteria have to be established through policy reuse (Fernández, García, and Veloso 2010), action correlation (Sherstov and Stone 2005), state abstraction (Walsh, Li, and Littman 2006), inter-space relation (Soni and Singh 2006), or other methods. Such mappings are hard to define when the agent operates in complex environments with large state spaces and multiple goal states, with possibly different state feature distributions and world dynamics. To efficiently accomplish varying tasks in heterogeneous environments, the agent has to learn to focus *attention* on the crucial features of each environment.

We propose a system that tries to transfer old knowledge, but at the same time evaluates new options to see if they work better. The agent gathers experience during its lifetime and enters a new environment equipped with *expectations* on how different aspects of the world affect the outcomes of agent's actions. The main idea is to allow an agent to collect a library of world models or representations, called *views*, that it can consult to focus its attention in a new task. In this dissertation, we concentrate on approximating the transition model. The reward model library can be learned in an analogous fashion. Effective utilization of the library of world models allows the agent to capture the transition dynamics of the new environment quickly; this should lead to a jumpstart in learning and faster convergence to a near optimal policy. The main challenge is learning to select a proper view for a new task in a new environment, without any predefined mapping strategies.

We will next formalize the problem and describe the method of collecting views into a library. We will then present an efficient implementation of the proposed transfer learning technique. After discussing related work, we will demonstrate the efficacy of our system through a set of experiments in two different domains.

6.1 TES: transferring expectations

A key idea of this approach is that the agent can represent the world dynamics from its sensory state space in different ways. Such different views correspond to the agent's decisions to focus attention on only some features of the state in order to quickly approximate the state transition function.

6.1.1 Decomposition of transition model

To allow knowledge transfer from one state space to another, we assume that each state s in all the state spaces can be *characterized* by a d-dimensional feature vector $f(s) \in R^d$. The states themselves may or may not be factored. In addition, state transitions are mediated via action effects. An action effect is more independent with specific states in a task' state space. We formulate a task by a CMDP$(S, f, A, T, E, R, \gamma)$, introduced in section 4.1.

For notation convenience, however, we re-define CMDP by a tuple $(S, A, E, \tau, \eta, f, R)$ in which the transition model is defined through the terms E, τ, η, f, where E is an effect set and f is a function from states to their feature vectors. $\tau : S \times A \times E \rightarrow [0, 1]$ is an action model such that $\tau(s, a, e) = P(e \mid f(s), a)$ indicates the probability of achieving effect e upon performing action a at state s. Notice that the probability of effect e depends on state s only through the features $f(s)$ in this formulation, though $f(s)$ may be defined to include state-attributes s as well. While the agent needs to learn the effects of the action, it is usually assumed to understand the meaning of the effects, i.e., how the effects turn each state into a next state. This knowledge is captured in a deterministic function $\eta : S \times E \rightarrow S$. Different effects e will change a state s to a different next state $s' = \eta(s, e)$. The MDP transition model T can be reconstructed from the CMDP by the equation:

$$T(s, a, s'; \tau) = P(s' \mid f(s), a) = \tau(s, a, e), \tag{6.1}$$

where e is the effect of action a that takes s to s', if such an e exists, otherwise $T(s, a, s'; \tau) = 0$.

6.1.2 A multi-view transfer framework

In our framework, the knowledge gathered and transferred by the agent is collected into a library $\mathcal{T}$ of online effect predictors or views.

A *view* consists of a structure component $\bar{f}$ that picks the features which should be focused on, and a quantitative component Θ that defines how these features should be combined to approximate the distribution of action effects. Formally, a *view* is defined as $\tau = (\bar{f}, \Theta)$, such that $P(E|S, a; \tau) = P(E|\bar{f}(S), a; \Theta) = \tau(S, a, E)$, in which $\bar{f}$ is an orthogonal projection of $f(s)$ to some subspace of R^d. Each view τ is specialized in predicting the effects of one action $a(\tau) \in A$ and it yields a probability distribution for the effects of the action a in any state. This prediction is based on the features of the state and the parameters $\Theta(\tau)$ of the view that may be adjusted based on actual effects observed in the task environment.

We denote the subset of views that specify effects for action a by $\mathcal{T}^a \subset \mathcal{T}$. The main challenge is to build and maintain a comprehensive set of views that can be used in new environments likely resembling the old ones, but at the same time allow adaptation to new tasks with completely new transition dynamics and feature distributions.

Algorithm 6 describes a procedure to build and use the library. At the beginning of every new task, the existing library is copied into a working library which is also augmented with fresh, uninformed views, one for each action, that are ready to be adapted to new tasks. We then select, for each action, a view with a good track record. This view is used to estimate the optimal policy based on the transition model specified in Equation 6.1, and the policy is used to pick the first action a. The action effect is then used to score all the views in the working library and to adjust their parameters. In each round the selection of views is repeated based on their scores, and the new

optimal policy is calculated based on the new selections. At the end of the task, the actual library is updated by possibly recruiting the views that have "performed well" and by retiring those that have not.

Algorithm 6 *TES*: **T**ransferring **E**xpectation**s** using a library of views

Input: $\mathcal{T} = \{\tau_1, \tau_2, ...\}$: view library; CMDP$_j$: a new j^{th} task;
Φ: view goodness evaluator

Let $\mathcal{T}_0$ be a set of fresh views - one for each action
$\mathcal{T}_{tmp} \leftarrow \mathcal{T} \cup \mathcal{T}_0$ /* THE WORKING LIBRARY FOR THE TASK */
for all $a \in A$ **do**
  $\hat{T}[a] \leftarrow \text{argmax}_{\tau \in \mathcal{T}^a} \Phi(\tau, j)$ /* SELECTING VIEWS */
end for
for $t = 0, 1, 2, ...$ **do**
  $a^t \leftarrow \hat{\pi}(s^t)$, where $\hat{\pi}$ is obtained by solving MDP using model $\hat{T}$
  Perform action a^t and observe effect e^t
  for all $\tau \in \mathcal{T}^{a^t}_{tmp} \cup \mathcal{T}^{a^t}$ **do**
    $Score[\tau] \leftarrow Score[\tau] + \log \tau(s^t, a^t, e^t)$
  end for
  for all $\tau \in \mathcal{T}^{a^t}_{tmp}$ **do**
    Update view τ based on $(f(s^t), a^t, e^t)$
  end for
  $\hat{T}[a^t] \leftarrow \text{argmax}_{\tau \in \mathcal{T}^{a^t}_{tmp}} Score[\tau]$ /* SELECTING VIEWS */
end for
for all $a \in A$ **do**
  $\tau^* \leftarrow \text{argmax}_{\tau \in \mathcal{T}^a_{tmp}} Score[\tau]$
  $\mathcal{T}^a \leftarrow growLibrary(\mathcal{T}^a, \tau^*, Score, j)$ /* UPDATING LIBRARY */
end for
if $|\mathcal{T}| > M$ **then**
  $\mathcal{T} \leftarrow \mathcal{T} - \{\text{argmin}_{\tau \in \mathcal{T}} \Phi(\tau, j)\}$ /* PRUNING LIBRARY */
end if

Scoring the views

To assess the quality of a view τ, we measure its predictive performance by a cumulative log-score. This is a *proper* score (Savage 1971) that can be effectively calculated online.

Given a sequence $D^a = (d^1, d^2, \ldots, d^N)$ of observations $d^i = (s^i, a, e^i)$ in which

action a has resulted in effect e^i in state s^i, the score for an a-specialized τ is

$$S(\tau, D^a) = \sum_{i=1}^{N} \log \tau\left(s^i, a, e^i; \theta(D^{:i-1}, \tau)\right),$$

where $\tau\left(s^i, a, e^i; \theta(D^{:i-1}, \tau)\right)$ is the probability of event e^i given by the event predictor τ based on the features of state s^i and the parameters $\theta(D^{:i-1}, \tau)$ that may have been adjusted using previous data $D^{:i-1} = (d^1, d^2, \ldots, d^{i-1})$. In section 6.2, we discuss ways to update θ incrementally.

Growing the library

After completing a task, the highest scoring new views for each action are considered for recruiting into the actual library. The winning new views are automatically accepted. In this case, the data has most probably come from the distribution that is far from the any current models, otherwise one of the current models would have had an advantage to adapt and win.

The winners τ^* that are adjusted versions of old views $\bar{\tau}$ are accepted as new members if they score significantly higher than their original versions, based on the logarithm of the prequential likelihood ratio (Dawid 1984) $\Lambda(\tau^*, \bar{\tau}) = S(\tau^*, D^a) - S(\bar{\tau}, D^a)$. Otherwise, the original versions $\bar{\tau}$ get their parameters updated to the new values. This procedure is just a heuristic and other inclusion and updating criteria may well be considered. The policy is detailed in Algorithm 7.

Pruning the library

To keep the library relatively compact, a plausible policy is to remove views that have not performed well for a long time, possibly because there are better predictors or they have become obsolete in the new tasks or environments. To implement such a retiring scheme, each view τ maintains a list H_τ of task indices that indicate the tasks for which the view has been the best scoring predictor for its specialty action $a(\tau)$. We

Algorithm 7 Grow sub-library $\mathcal{T}^a$

Input: $\mathcal{T}^a$, τ^*, $Score$, j: task index; c: constant;
$H_{\tau^*} = \{\}$: empty history record
Output: updated library subset $\mathcal{T}^a$ and winning histories H_{τ^*}

if $\tau^* \in \mathcal{T}_0^a$ **then**
  $\mathcal{T}^a \leftarrow \mathcal{T}^a \cup \{\tau^*\}$ /* ADD NEWBIE TO LIBRARY */
else
  Let $\bar{\tau} \in \mathcal{T}$ be the original, not adapted version of τ^*
  if $Score[\tau^*] - Score[\bar{\tau}] > c$ **then**
    $\mathcal{T}^a \leftarrow \mathcal{T}^a \cup \{\tau^*\}$
  else
    $\mathcal{T}^a \leftarrow \mathcal{T}^a \cup \{\tau^*\} - \{\bar{\tau}\}$
    $H_{\tau^*} \leftarrow H_{\bar{\tau}}$ /* INHERIT HISTORY */
  end if
end if
$H_{\tau^*} \leftarrow H_{\tau^*} \cup \{j\}$

can then calculate the recency weighted track record for each view. In practice, we have adopted the procedure by Zhu et al. (Zhu, Ghahramani, and Lafferty 2005) that introduces the recency weighted score at time T as

$$\Phi(\tau, T) = \sum_{t \in H_\tau} e^{-\mu(T-t)},$$

where μ controls the speed of decay of past success. Other decay functions could naturally also be used. The pruning can then be done by introducing a threshold for recency weighted score or always maintaining top M views.

6.2 View learning

In *TES*, a view can be implemented by any probabilistic classification model that can be quickly learned online. We employ *mDAGL* (see section 4.2), an online multinomial logistic regression algorithm with group Lasso. While multinomial logistic regression models cannot capture all the conditional distributions, their simplicity allows fast online learning in very high dimensional spaces. The detailed analyses

in Chapter 4 and 5 have demonstrated the efficiency of this approach.

Although learning views is necessary for an autonomous agent, in some applications this library of potential views may be made readily available by domain experts. The agent is left with only selecting and adapting appropriate views for an environment. Nguyen et al. (2012a) have implemented another simplified version of *TES* in which a view is predefined with a DBN structure. View parameters are modeled by Dirichlet distribution that is updated by Bayes rule.

6.3 Experiments

We examine the performance of our expectation transfer algorithm *TES* that transfers views to speed-up the learning process across different environments in two benchmark domains. We show that *TES* can efficiently: a) learn the appropriate views online, b) select views using the proposed scoring metric, c) achieve a good jump start, and d) perform well in the long run.

To better compare with some related work, we evaluate the performance of *TES* for transferring both transition models and reward models in RL. *TES* can be adapted to transfer reward models as follows: Assuming that the rewards follow a Gaussian distribution, a view of the expected reward model can be learned similarly as in section 6.2. We use an online sparse linear regression model instead of the multinomial logistic regression. Simply replacing matrix W by a vector w, and using squared loss function, the coefficient update function can be found similar to that in Equation 4.3 (Xiao 2009). When studying reward models, the transition models are assumed to be known.

6.3.1 Learning views for effective transfer

In the first experiment, we compare *TES* with the locally weighted *LWT* approach by Atkeson et al. (Atkeson, Moore, and Schaal 1997) and with the non-parametric hierarchical Bayesian approach *HB* by Wilson et al. (Wilson et al. 2007) in transfer-

ring reward models. We adopt the same domain as described in Wilson et al.'s *HB* paper, but augment each state with 200 random binary features. The objective is to find the optimal route to a known goal state in a color maze. Assuming a deterministic transition model, the highest cumulative reward, determined by the colors around each cell/state, can be achieved on the optimal route.

Experiment set-up: Five different reward models are generated by normal Gaussian distributions, each depending on different sets of features. The start state is random. We run experiments on 15 tasks repeatedly 20 times, and conduct leave-one-task-out test. The maximum size M of the views library, that is initially empty, is set to be 20; threshold c for growing the library is set to be $\log 300$. The parameters for view learning are: $\lambda = 0.05$ and $\alpha = 2.5$.

Table 6.1: Transfer of reward models: Cumulative reward in the first episodes; Time to solve 15 tasks (in minutes), in which each is run with 200 episodes. Map sizes vary from 20×20 to 30×30.

Methods	Tasks															**Time**
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
HB	-108.01	-85.26	-67.46	-90.17	-130.11	**-95.42**	-46.23	-77.10	**-83.91**	**-51.01**	-131.44	-97.05	-90.11	**-48.91**	-92.31	77.2
LWT	-79.41	-114.28	-83.31	**-46.70**	-245.11	-156.23	-47.05	**-49.52**	-105.24	-88.19	-174.15	-85.10	**-55.45**	-101.24	-86.01	28.6
TES	**-45.01**	**-78.23**	**-62.15**	-54.46	**-119.76**	-115.77	**-37.15**	-58.09	-167.13	-59.11	**-102.46**	**-45.99**	-86.12	-67.23	**-81.39**	31.5

As seen in Table 6.1, *TES* on average wins over *HB* in 11 and *LWT* in 12 out of 15 tasks. In the $15 \times 20 = 300$ runs *TES* wins over *HB* 239 and over *LWT* 279 times, both yielding binomial test p-values less than 0.05. This demonstrates that *TES* can successfully learn views and utilize them in novel tasks. Moreover, *TES* runs much faster than *HB*, and just slightly slower than *LWT*. Since *HB* does not learn the relevant features for model representation, it may overfit, and the knowledge learned cannot be easily generalized. It also needs a costly sampling method. Similarly, the strategy for *LWT* that tries to learn one common model for transfer in various tasks often does not work well.

6.3.2 Multi-view transfer in complex environments

In the second experiment, we evaluate *TES* in a more challenging domain, transferring transition models. We consider a grid-based robot navigation problem in which each grid-cell has the surface of either sand, soil, water, brick, or fire. In addition, there may be walls between cells. The surfaces and walls determine the stochastic dynamics of the world. However, the agent also observes numerous other features in the environment. The agent has to learn to focus on the relevant features to quickly achieve its goal. The goal is to reach any exit door in the world consuming as little energy as possible.

Experiment set-up: The agent can perform four actions (move up, down, left, right) which will lead it to one of the four states around it, or leave it to its current state if it bumps into a wall. The agent will spend 0.01 units of energy to perform an action. It loses 1 unit if falling into a fire, but gains 1 unit when reaching an exit door. A task ends when the agent reaches any exit door or fire.

We design fifteen tasks with grid sizes ranging from 20×20 to 30×30. Each task has a different state space and different terminal states. Each state (cell) also has 200 irrelevant random binary features, besides its surface materials and the walls around it. The tasks may have different dynamics as well as different distributions of the surface materials. In our experiments, the environment transition dynamics is generated using three different sets of multinomial logistic regression models (Appendix D) so that every combination of cell surfaces and walls around the cell will lead to a different transition dynamics at the cell. The probability of going through a wall is rounded to zero and the freed probability mass is evenly distributed to other effects. The agent's starting position is randomly picked in each episode.

We represent five effects of the actions: moved up, left, down, right, did not move. The maximum size M of the view library, initially empty, is set to be 20; threshold $c = \log 300$. In a new environment, the *TES*-agent mainly relies on its transferred knowledge. However, we allow some ϵ-greedy exploration with $\epsilon = 0.05$. The

parameters for view learning algorithm are that $\lambda = 0.05$, $\alpha = 1.5$.

We conduct leave-one-out cross-validation experiment with fifteen different tasks. In each scenario the agent is first allowed to experience fourteen tasks, over 100 episodes in each, and it is then tested on the remaining one task. No recency weighting is used to calculate the goodness of the views in the library. We next discuss experimental results averaged over 20 runs showing 95% confidence intervals for some representative tasks.

Transferring expectations between homogeneous tasks

To ensure that *TES* is capable of basic model transfer, we first evaluate it on a simple task to ensure that the learning algorithm in section 6.2 works. We train and test *TES* on two environments which have same dynamics and 200 irrelevant binary features that challenge agent's ability to learn a compact model for transfer. Figure 6-1 shows how much the other methods lose to *TES* in terms of accumulated reward in the test task. *loreRL* is an implementation of *TES* equipped with the view learning algorithm that does not transfer knowledge. *fRmax* is the factored *Rmax* (Brafman and Tennenholtz 2002) in which network structures of transition models are provided by an oracle (Strehl, Diuk, and Littman 2007); its parameter m is set to be 10 in all experiments. *fEpsG* is a heuristic in which the optimistic *Rmax* exploration of *fRmax* is replaced by an ϵ-greedy strategy ($\epsilon = 0.1$). The results show that these oracle methods still have to spend time for learning the parameters of their models, so they gain less accumulated reward than *TES*. This also suggests that the transferred view of *TES* is likely not only compact but also accurate. Figure 6-1 further shows that *loreRL* and *fEpsG* are more effective than *fRmax* in early episodes.

View selection vs. random views

Figure 6-2 shows how different views lead to different policies and accumulated rewards over the first 50 episodes in a given task. The *Rands* curves show the accumu-

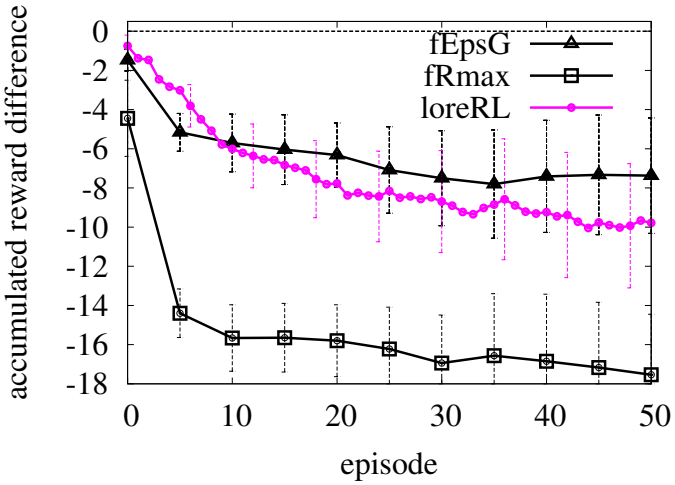


Figure 6-1: Performance difference to *TES* in early trials in homogeneous environments.

lated reward difference to *TES* when the agent follows some random combinations of views from the library. For clarity we show only 5 such random combinations. For all these, the difference turns negative fast in the beginning indicating less reward in early episodes. We conclude that our view selection criterion outperforms random selection.

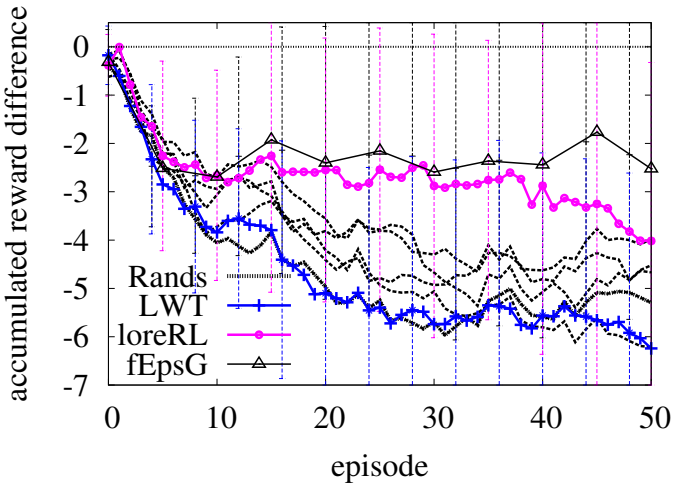


Figure 6-2: Performance difference to *TES* in early trials in heterogeneous environments.

Table 6.2: Cumulative reward after first episodes. For example, in Task 1 *TES* can save $(0.616 - 0.113)/0.01 = 50.3$ actions compared to *LWT*.

Methods	Tasks														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
loreRL	-0.681	-0.826	-0.814	-1.068	-0.575	-0.810	-0.529	-0.398	-0.653	-0.518	-0.528	-0.244	-0.173	-1.176	-0.692
LWT	0.113	-0.966	-0.300	**0.024**	-1.205	**-0.345**	-1.104	-1.98	-0.057	-0.664	**-0.230**	-1.228	0.034	0.244	-0.564
TES	**0.616**	**-0.369**	**0.230**	-0.044	**-0.541**	-0.784	**-0.265**	**0.255**	**0.001**	**-0.298**	-1.184	**-0.077**	**0.209**	**0.389**	**-0.407**

Multiple views vs. single view, and non-transfer

We compare the multi-view learning *TES* agent with a non-transfer agent *loreRL*, and an *LWT* agent that tries to learn only one good model for transfer. We also compare with the oracle method *fEpsG*. As seen in Figure 6-2, *TES* outperforms *LWT* which, due to differences in the tasks, also performs worse than *loreRL*. When the earlier training tasks are similar to the test task, the *LWT* agent performs well. However, the *TES* agent also quickly picks the correct views, thus we never lose much but often gain a lot. We also notice that *TES* achieves a higher accumulated reward than *loreRL* and *fEpsG* that are bound to make uninformed decisions in the beginning.

Table 6.2 shows the average cumulative reward after the first episode (the jumpstart effect) for each test task in the leave-one-out cross-validation. We observe that *TES* usually outperforms both the non-transfer and the *LWT* approach. In all $15 \times 20 = 300$ runs, *TES* wins over *LWT* 247 times and it wins over *loreRL* 263 times yielding p-values smaller than 0.05.

We also notice that due to its capability of fast capturing the world dynamics, *TES* running time is just slightly longer than *LWT*'s and *loreRL*'s, which do not perform extra work for view switching but need more time and data to learn the dynamics models.

Convergence

To study the asymptotic performance of *TES*, we compare with the oracle method *fRmax* which is known to converge to a (near) optimal policy. Notice that in this feature-rich domain, *fRmax* without the pre-defined DBN structure is just similar to

Rmax. Therefore, we also compare with *Rmax*. For *Rmax*, the number of visits to any state before it is considered "known" is set to 5, and the exploration probability ϵ for known states starts to decrease from value 0.1.

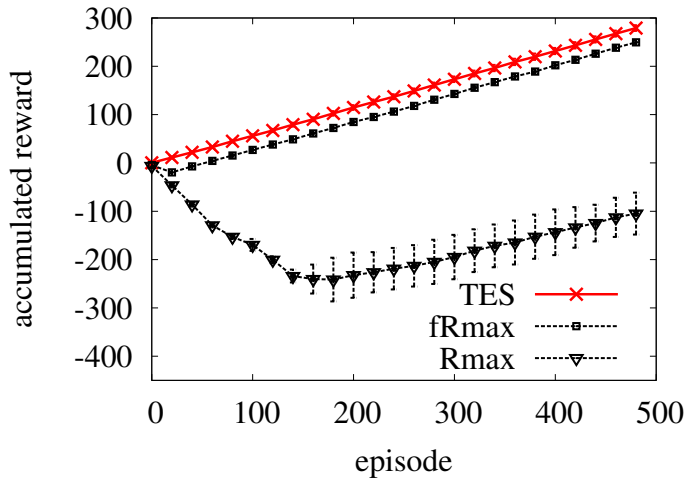


Figure 6-3: Asymptotic performance.

Figure 6-3 shows the accumulated rewards and their statistical dispersion over episodes. Average performance is reflected by the angles of the curves. As seen, *TES* can achieve a (near) optimal policy very fast and sustain its good performance over the long run. It is only gradually caught up by *fRmax* and *Rmax*. This suggests that *TES* have succesfully learned a good library of views in heterogneous environments and efficiently utilizes those views in novel tasks.

6.4 Discussion

We have presented a framework for learning and transferring multiple expectations or views about world dynamics in heterogeneous environments. Not surprisingly, when the environments are different, the combination of learning multiple views and dynamically selecting the most promising ones yields a system that can learn a good policy faster and gain higher accumulated reward compared to the common strategy of learning just a single good model and using it in all occasions.

Multiple models have previously been used to guide behavior in non-stationary environments (Doya et al. 2002) (Silva et al. 2006). Unlike our work, these studies usually assume a common concrete state space. In representation selection, Konidaris and Barto (Konidaris and Barto 2009) focus on selecting the best abstraction to assist agent's skill learning, and Van et al. (Van Seijen, Bakker, and Kester 2008) study using multiple representations together to solve an RL problem. None of these studies, however, solve the problem of transferring knowledge in heterogeneous environments.

Utilizing and maintaining multiple models requires additional computation and memory. We have shown that by a clever decomposition of the transition function, the model selection and model updating can be accomplished efficiently using online algorithms. Our experiments demonstrate that performance improvements in multi-dimensional heterogeneous environments can be achieved with a small computational cost.

6.5 Summary

We have introduced *TES*, a novel framework of transferring expectations using a library of views, to transfer knowledge in heterogeneous environments. We have also described a detailed combination of *loreLR* and *TES* to build a life-long learning agent as sketched in Chapter 3. The agent could learn, accumulate, and transfer knowledge in various tasks. In the next chapter, we show case-studies evaluating the proposed theoretical frameworks in a real robotic domain.

Chapter 7

Case-studies: working with a real robotic domain

We have seen theoretical analyses showing the advantages of *loreRL* and *TES* over the state of the art model-based RL algorithms. On simulated domains, various empirical evaluations have also confirmed the efficiency of our methods. The results, though valuable, are obtained under assumptions that are favorable for our approach. In this chapter, we aim to further evaluate the proposed methods on a real robotic domain where we cannot expect the effect of actions to follow a logistic regression model. The content in this chapter has been partly published in Proceedings of the International Conference on Machine Learning (ICML'13) in 2013 (Nguyen et al. 2013).

7.1 Environments

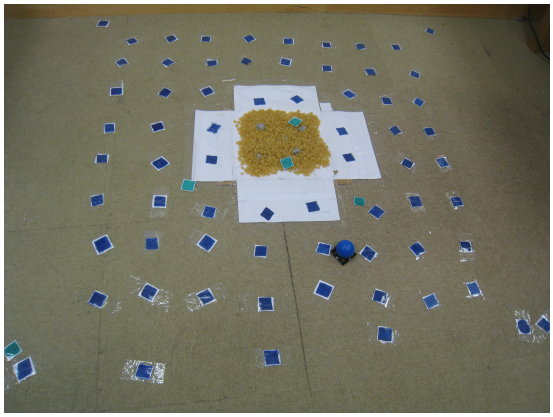
Figure 7-1 shows all the three environments used in our case studies. They are designed so that the robot's actions would have different effects at different locations, and the environment surfaces are main factors affecting the action effects. The surfaces

are made of various materials such as beans, soil, hay, leaves, shells, paper board, and nylon Berber carpet. These materials have different physical effects on the objects that are moving on them. The slopes and obstacles on the surfaces also contribute to the effects of the actions. In some areas, the surfaces may change because of the robot's actions. We repair the surfaces to the original condition after every episode.

For a robot to efficiently plan its path in these environments, one would need to carefully design and extract a small set of features based on the slopes, the obstacles, and the materials of the surfaces in different areas of the environments. However, that task is usually very difficult. It is preferable to leave the robot to automatically select relevant features from a large set of simple features by itself. To test our approach, we, therefore, simply draw on the surfaces green and blue marks. The robot is also marked with two red marks. There are also a blue ball, and several death-marked spots in the environment. These things can form very many features. The robot will need to select a few features that may serve as proxies to the true factors that affect its action effects.

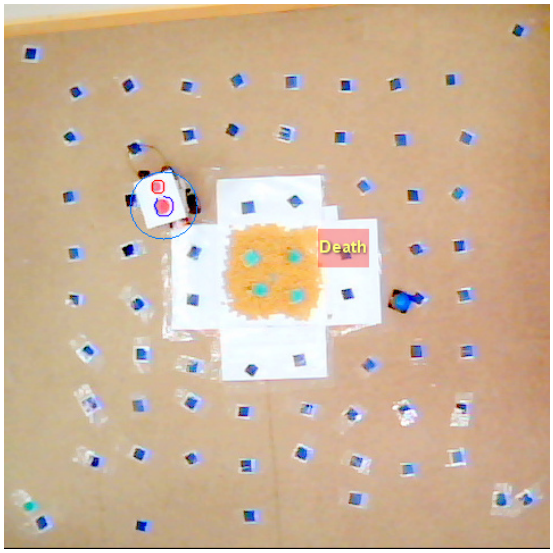
Environment 1 and 3 are deliberately designed so that the robot should base its *views* (transition model) on the blue marks. The transition dynamics models in these two environments are very similar. However, the two environments have also differences (in irrelevant features): the blue balls, the death places, and the green marks are at different locations. In section 7.2.3, we will explain the features. Environment 2 is very different from Environment 1 and 3. It is designed so that the robot should base its views on the green marks instead of the blue ones.

We treat the environments as discrete MDPs. We discretize the Environment 2 into a state space consisting of 8×8 (x,y)-locations and 8 different orientations of a robot, which yields a state space of 512 states. Environment 1 and 3 are larger, so we discretize them into a state space consisting of 10×10 (x,y) locations and 8 different orientations of a robot. The environments are in two different sizes, 5×5 feet and 6×6 feet (Figure. 7-1).



(a) Environment 1: 6×6 feet.

(b) Environment 2: 5×5 feet.



(c) Environment 3: 6×6 feet.

Figure 7-1: Three different environments.

7.2 Robot

We use the LEGO Mindstorms NXT v1.1 kit to build a three-wheel robot depicted in Figure 7-2. Two front wheels of the robot are attached to two separate motors; the back wheel is free rolling. The track width is 11.2 cm. The robot carries a white panel on top with a big and a small red marks for the positioning system to locate the robot's position and orientation.

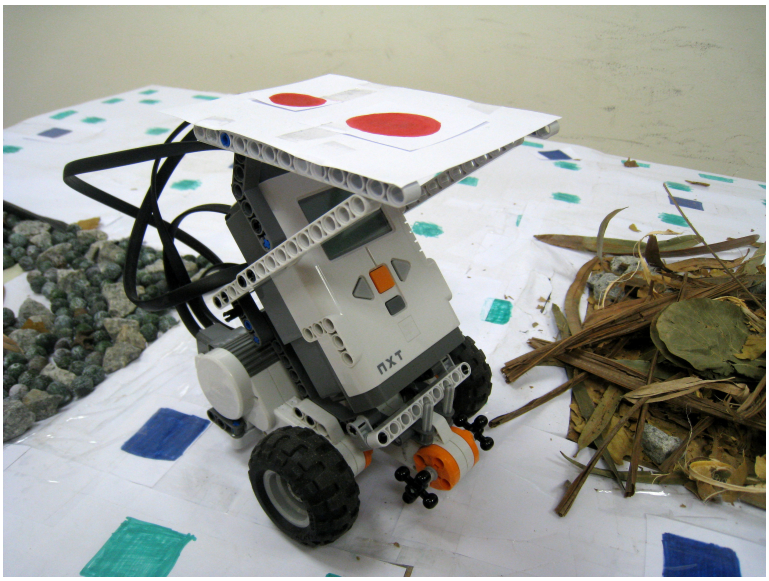


Figure 7-2: The robot.

The whole robot system is comprised of three main components, including a central processor, an observatory system, and a command controller. The positioning system is a sub-component of the observatory system. Information on the environment and the robot's position is captured by a webcam and sent from the *observatory system* to the *central processor* to update robot's knowledge-base as well as to plan the next action. The webcam is attached to the ceiling above the area so that the robot can fully observe the environment. The action command is then transmitted via Bluetooth to the *command controller* embedded in the robot to execute. We implement the controller in leJOS [1].

[1] leJOS, Java for LEGO Mindstorms. http://lejos.sourceforge.net/

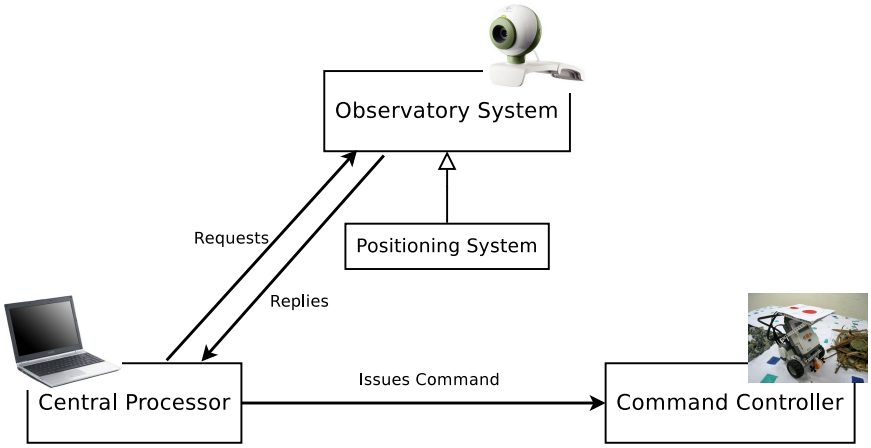


Figure 7-3: The system architecture.

7.2.1 Actions

The robot is programmed to rotate its left and right wheels in three different ways corresponding to three actions. For the first action, the robot rotates both its left and right wheels 246 degrees. For the second action, the robot rotates its left wheel 90 and right wheel -90 degrees at the same time. For the third action, the robot rotates its left wheel -90 and right wheel 90 degree. As the robot may be still moving after each action, we let the system idle for 200 milliseconds after an action waiting for the robot to stop completely. These actions, under ideal situations, correspond to the actions of move-forward, turn-left, and turn-right, respectively.

Action effects

Due to inaccurate robot motors, sensors, and various real world factors like the surface materials, slopes of the surface, and obstacles, the actions may change the robot's relative location in four different ways, including moved forward one cell, moved diagonally forward to the cell on the robot's left, moved diagonally forward to the cell on the robot's right, and did not move. The robot's orientation can also be changed in five different ways, including: turned to the next orientation on left, the second next orientation on left, the next orientation on right, the second next orientation on right,

and did not turn. That would result in a total of 20 different effects.

7.2.2 Sensor

We mainly process information only from the web-cam in the observatory system. The web-cam is attached to the ceiling above the area. The robot, therefore, can fully observe the environment. However, the robot can only capture the big and small red marks on the top of the robot itself, and the information of the locations of the green, blue, red marks, and the ball in the environment. As the features are simple, we just use basic algorithms in OpenCV library [2] to detect them. The result is nearly perfect.

7.2.3 Factorization: state-attributes and state-features

As mentioned, an environment is discretized to $n\ rows \times m\ columns \times\ o\ orientations$, so the full environment state space can be identified or factorized by those three *state attributes*. However, these attributes alone do not contain enough information for predicting an action effect or transition dynamics. Therefore, it is critical that the agent also describes each state with a long vector of binary *state features*. The "green" binary indicator $f_i^G(s)$ of a state s is set to 1 iff there is a green mark that is further than i units but closer than $i + 1$ units from the xy-center of the state s ($i \in \{0, \ldots, 99\}$). A unit equals the width of the environment divided by 100. Similar features are defined for blue marks and to the blue target ball yielding 300 binary features. Eight indicators for different robot orientations are also included in the feature-base together with four intentionally redundant "there is/is-not a green/blue mark in a state"-bits. All together these yield 312 binary features per state. The intuition behind these features is that they serve as proxies to surface materials, slopes on the surfaces, obstacles, etc. which appear to be important factors determining the dynamics in the environments, but the robot's sensors cannot capture. Although only few among these 312 features are

[2] http://opencv.org/

important for modeling robot's actions, the robot does not know which ones. The robot has to learn to select them based on feedback while interacting with the environment.

7.3 Task

The robot is assumed to know the reward model before any start. The robot's task is to travel in the environments from a random starting point to reach the *blue ball*, which will earn it a reward of 2 points. The robot will receive -1 point if it falls out of the area or into death places marked with orange rectangles, and -0.05 points for an action at any other states. An episode ends if the robot reaches a terminal state, or gets stuck for four consecutive actions.

In other words, the robot aims for the highest cumulative reward in each episode. It tries to reach the blue ball as fast as possible, but avoid the death places or being fallen out of the map. In case it is very costly or impossible to reach the ball, the robot could give up by heading out of the map or running into a death place.

7.4 Experiments

We will first evaluate our new model-based RL algorithm, *loreRL*, on Environment 2 (Figure 7-1b) to understand the practical qualities of our new state factorization, as well as the use of simple multinomial logistic regression models in representing the world transition dynamics. We then test the *TES* framework, in which *loreRL* is a component, to see if *TES* could effectively transfer knowledge in heterogeneous environments in practice.

7.4.1 Evaluation of *loreRL*

The robot battery do not allow us to compare our algorithm with the slow *RL-DT* and *LSE-Rmax* algorithms, thus we will only compare with the fine-tuned algorithms,

including *fRmax*, *fEpsG*, and *man-loreRL*, in which we manually try to select important features and specify the DBN-structures for the transition models. *man-loreRL* is based on multinomial logistic regression models with the manually selected 12 important features, including eight indicators for different robot orientations, and four indicators telling if there is/is-not a green/blue mark in a state. We run the experiments with *loreRL* and *man-loreRL* setting $\alpha = 0.5$, $\lambda = 0.05$, $\gamma = 0.95$, exploration $\epsilon = 0.05$, and parameter $m = 10$ for *fRmax*. All results are averaged over 20 runs, and we report the 95% confidence intervals.

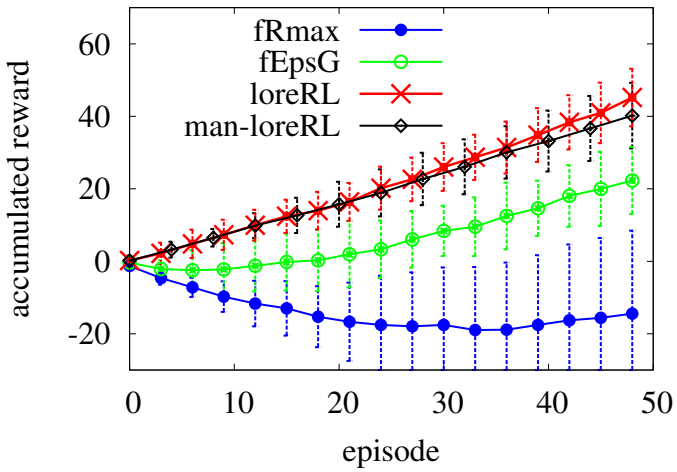


Figure 7-4: Accumulated rewards by various methods.

As shown in Figure 7-4, *loreRL* appears to quickly capture the environment dynamics and outperform other methods. Even with manually selected features, *fRmax* and *fEpsG* require more exploration to learn the dynamics. *man-loreRL* gains the rewards a bit faster, but in the end it loses to *loreRL* slightly, possibly due to the (unforeseen) insufficiency of the manually selected features. Table 7.1 further shows that *loreRL* is fast. Its average running time per episode with 312-features is only slightly slower than with 12 manually selected features.

The evaluations above have demonstrated that our new model-based RL, *loreRL* may work effectively in the real world. It may converge fast to a near optimal policy, and so achieve a high accumulated reward.

Table 7.1: Average running time per episode in 50 episodes. Run on Intel Centrino Duo T2400 (1.83GHz), 2 × 512 MB RAM.

Algorithm	*fRmax*	*fEpsG*	*man-loreRL*	***loreRL***
Time (sec.)	13.44	12.77	9.35	10.81

7.4.2 Evaluation of *TES*

This case-study is designed to test if *TES* could effectively manage a good library of views to reduce the negative transfer effects and achieve better performance than other methods. We compare the robot's performance in four scenarios as detailed below, and report the results for accumulated reward, jumpstart, and running time.

1. *loreRL*: the robot has no experience in Environments 1 and 2. It runs and learns views (transition model) directly on Environment 3.

2. *LWT*: the robot has first experienced Environment 2, and then uses that knowledge (transition model) to learn action policy and run on Environment 3. The robot does not update its knowledge of the transition model in the new environment.

3. *TES-woRi*: the robot has first experienced Environment 2, and then runs on Environment 3. Different from the scenario 2, however, the robot here has the capability to adapt and develop new transition model for this novel Environment 3 if the robot sees that is necessary. This *TES-woRi* robot is actually the *TES* robot, but we do not let it to experience the environment that is similar to the testing environment. This set-up evaluates *TES*'s capability to work with novel environments and to reduce negative transfer effects.

4. *TES*: the robot has first experienced Environment 1, and then Environment 2, before it runs on Environment 3. By this setting, we will see how effectively *TES* builds the library, and selects views from the library to solve a new task.

Table 7.2 summarizes and highlights the differences in these four scenarios.

Table 7.2: Four robot testing scenarios.

Scenarios	Transferring knowledge	Experiences	Developing new views	Test on
1. *loreRL*	No	No	Yes	Env. 3
2. *LWT*	Yes	Env. 2	No	Env. 3
3. *TES-woRi*	Yes	Env. 2	Yes	Env. 3
4. *TES*	Yes	Env. 1 and Env. 2	Yes	Env. 3

We do not test the setting of letting the robot to experience with Environment 1 and running on Environment 3. This setting would allow us to see if *TES* can learn good views to transfer between similar Environment 1 and 3. However, this effect can also be observed clearly by comparing *TES* and *TES-woRi*. If *TES* outperforms *TES-woRi*, it means that *TES* learnt compact views of Environment 1 into the library.

Results for accumulated reward

Figure 7-5 shows the differences in accumulated rewards of the robot with *LWT*, *TES-woRi*, and *loreRL* respectively to the robot with *TES*. As seen in the figure, the differences are all below 0, suggesting that *TES* could effectively transfer the view library in heterogeneous environments. *TES* could select the best models to approximate the world dynamics quickly, and outperform the other methods. It also suggests that *TES* with *mDAGL* has successfully learnt compact action models, likely without redundant features, to the view library for transfer. For example, the robot originally has 100 features related to the ball position, and the balls in environment 1 and 3 are at different positions, but the robot can still take advantage of the transferred knowledge. Checking the transferred model also confirms our hypothesis.

In addition, the data shows that *LWT* achieves the lowest accumulated reward, suggesting that *LWT* has serious negative transfer effect, which usually appears when knowledge is transferred in heterogeneous environments. *TES-woRi* also suffers from the negative transfer effect, but due to its ability to quickly recognize the unsuitability of the transferred knowledge, it could adopt new action models to alleviate the negative effect. We see that *TES-woRi* loses to *loreRL* but not to a large extent.

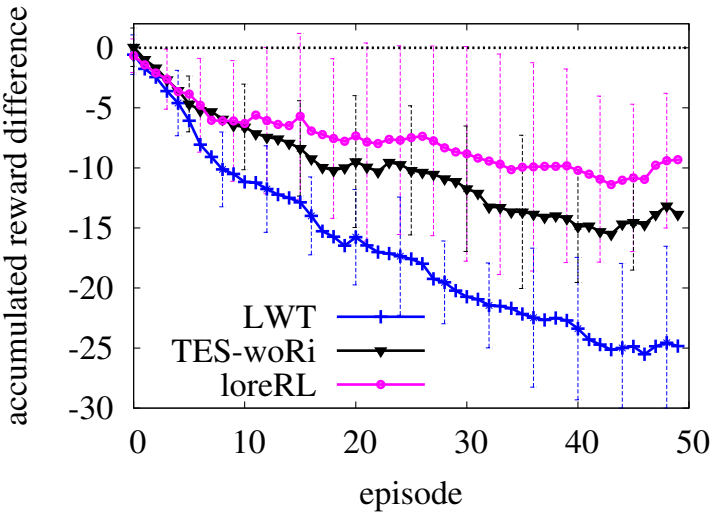


Figure 7-5: Performance difference to *TES* in early trials in robotic domain.

Table 7.3: The robot cumulative rewards after the first episodes in 10 repeats.

Method	**Repeat**										
	Avg	1	2	3	4	5	6	7	8	9	10
loreRL	-0.35	0.4	-2.45	-0.9	0.65	-1	-0.8	0.7	0.25	0.5	-0.85
LWT	-0.26	1.45	-1.1	-0.55	-0.5	-0.45	1.4	-0.5	-2.4	1	-0.95
TES-woRi	**0.36**	-0.85	-0.5	1.3	1.1	-0.6	0.3	1.05	-0.75	1.45	1.1
TES	**0.31**	0.15	0.8	-0.4	0.95	1.3	0.8	0.75	0.85	-1.15	-0.95

Results for jumpstart

To examine the jumpstart, we organize the cumulative rewards after the first episodes in 10 repeat for all four algorithms into Table 7.3. In 10 repeating runs *TES* wins over *loreRL* 7 times, *LWT* 6 times (draw in 1 repeat), *TES-woRi* 5 times; *TES-woRi* wins over *loreRL* 8 times, *LWT* 7 times. We see that the *TES* methods have lower rewards than the others in several runs, and do not statistically significantly show superiorities over the others. The result, however, is as anticipated, because *TES* requires explorative interactions with the environment to select the appropriate views, but the starting locations in this test are quite close to the terminal "death" states. Nevertheless, it should be noted that *TES* methods still have better performance than the others in more runs. In addition, on average they gain higher jumpstarts.

Table 7.4: The robot's average running time per episode in 50 episode runs. The systems are run on an Intel Core Duo Processor T2400 (1.83 GHz) laptop with 2 × 512 MB RAM.

Method	*loreRL*	*LWT*	*TES-woRi*	*TES*
Time (sec.)	18.32	13.13	14.34	16.23

Results for running time

Table 7.4 shows the robot's running times per episode averaged over 50 episodes. A reason that *LWT* has the shortest average running time is that *LWT* transfers and uses only one model for each action. In addition, since *mDAGL* is used in the implementation of *LWT*, the transferred model is quite compact with only a few features and so can be learnt quickly. Besides, the policy learnt based on that "wrong" model appears to guide the robot to the terminal "death" states or to go out of the map, so the episodes run with *LWT* is often shorter than the others.

TES-woRi and *TES* have to manage a larger view library of 3×2 and 3×3 action models respectively, so their running times are longer. However it is interesting to note that they are not much slower than *LWT*, and even faster than *loreRL* which does not spend time to process transferred knowledge. That is probably because *TES* can save quite a lot of time from not having to plan an optimal policy with complex action models. The *TES* methods start the planning with compact and simple transferred action models, and switch to use fresh models later, if necessary, after the robot has accumulated some data to eliminate a large number of noncritical features in the model representations. Of course, another possible reason may be that the libraries in this experiment are relatively small.

7.5 Discussion

The two series of experiments in the robotic domain have convincingly shown that our new model-based RL algorithm, *loreRL*, built upon CMDP, the new way of state fac-

torization, and *mDAGL*, the sparse multinomial logistic regression, may be effective in solving complex and feature-rich problems. The *TES* could be an efficient framework for transferring knowledge in heterogeneous environments. We conclude that it is possible to construct a smart autonomous robot which can select features, accumulate and transfer knowledge in real environments by integrating the *loreRL* into the *TES*.

There are several issues that one needs to be concerned with when extending the proposed methods and using them for real world applications. While running the experiments, we see that *loreRL* could not always follow an optimal policy because we could not fully satisfy the i.i.d data sampling requirement for *mDAGL*. *mDAGL* may quickly find a model to approximate the transition dynamics. After that the robot mostly, however, "sees" data in only some limited trajectories since it closely follows the policy constructed by the model. Consequently, the robot may update its transition model farther from the true dynamics. To mitigate this effect, the robot may have to temporarily use a less efficient policy for some periods of time. Nevertheless, in an ever uncertain world, this issue is not very critical as the robot has to do exploration for better policies quite often anyway.

We have also limited the analyses of our methods to the robot with a discrete action space. A robot's action is usually a result of a combination of various continuous parameters. However, in this study we manually specified just a set of particular actions. Consequently, the robot could not fully utilize all of its functional capabilities to learn a sophisticated policy so as to work efficiently in the real environments. For example, when the robot is 2 inches from the ball, technically the robot should be able to control its motors to move 2 inches forward to reach the ball. Perhaps, a simple way to address this continuous action issue may be to encode action parameters as features into those regression models. In our customized DBNs of robot's actions, each network local structure is a generalized regression model whose input is just a vector of features.

In addition, we attached a camera to the ceiling so that the robot could always

capture the whole environment to plan its actions. This setting may not be satisfied in many applications where the robot has to operate in outdoor environments, for example. However, this constraint is not because of our proposed methods, but rather due to the policy learning algorithm, value iteration. We could have used other methods instead, such as *Dyna-Q* (Sutton 1990) or *Prioritized Sweeping* (Moore and Atkeson 1993), but we chose this basic algorithm to better highlight our contributions to the learning and transferring of transition (and reward) models.

Chapter 8

Conclusion and future work

The objective of this dissertation is to build a life-long model-based RL agent that could automatically and efficiently learn, and transfer knowledge between tasks based mainly on feedback from their environments. The environments may be complex and feature-rich. In addition, they may be heterogeneous – the transition, reward dynamics, feature distributions, state spaces, or terminal states in the environments of different tasks may be very different.

8.1 Summary and conclusion

Compared to the state of the art methods in both theoretical and empirical evaluations, the proposed methods have shown to be more efficient in finding a (near) optimal policy, gaining a higher jumpstart and accumulated reward in a task.

We proposed CMDP, a variant formulation of the factored MDP that incorporates a principled way to compactly factorize the state space, while capturing comprehensive transition and reward dynamics information; we distinguished between state-attributes and state-features, and constructed the transition model through predicting how the

actions change the world. We also proposed *mDAGL*, a simple and efficient online multinomial logistic regression method with group lasso to automatically learn the relevant structure as well as parameters of the transition model. In other words, the system would incrementally learn to focus its attention on just a few relevant features while ignoring many redundant features to form expectations approximating the world dynamics. The regression models cannot capture the full conditional distributions like DBNs, but their simplicity allows fast, online learning in very high dimensional spaces. These techniques appear vital for an autonomous agent to operate in real environments.

In addition, we introduced *loreRL*, a model-based RL with online feature selection. In *loreRL*, we demonstrated how online multinomial logistic regression with group lasso can be used to quickly obtain a parsimonious transition model in model based RL. The method leads to fast learning since a single transition model can be learnt using samples from all the states with a small set of features. Unlike the other recent methods based on DBN (Kearns and Koller 1999; Degris, Sigaud, and Wuillemin 2006; Strehl, Diuk, and Littman 2007; Ross and Pineau 2008; Diuk, Li, and Leffler 2009; Hester and Stone 2009; Chakraborty and Stone 2011; Hester and Stone 2012), our method may be less generalized to accurately represent all transition models. However, this trade-off between generality and scalability may be useful in various complex real world applications.

Further, we presented *TES*, an efficient and simple method for learning and transferring expectations using a library of views about the world dynamics in heterogeneous environments. No manual state mappings or assumptions on the similarities between environments are required. Incorporating *loreRL* into *TES*, we could, consequently, achieve a unified system that could learn, accumulate, and transfer knowledge in complex, feature-rich, and heterogeneous environments. The experimental scenarios suggested that when the environments are different, the combination of learning multiple views and dynamically selecting the most promising ones yields a system

that can learn a good policy faster and gain higher accumulated reward compared to the recent methods (Atkeson, Moore, and Schaal 1997; Wilson et al. 2007). In a long run over very many episodes and tasks, the hierarchical Bayesian approach by Wilson et al. may find better policies than *TES*, but that approach has high computational cost and is slow.

Last but not least, case-studies on a real robotic domain showed that the assumptions of i.i.d. data observation, multinomial logistic distributions and action effects are not very detrimental to the performance of the autonomous robot in practice. It appears that the proposed model-based RL system is actually able to select features online to model the world dynamics, and transfer the views of these models to operate and adapt effectively in different, unfamiliar real environments. Table 8.1 briefly constrasts the proposed methods with the existing related work.

Table 8.1: A summary of important methods discussed in this work. We describe our methods and the highly related ones: RL-DT (Hester and Stone 2009), Met-Rmax (Diuk, Li, and Leffler 2009), LSE-Rmax (Chakraborty and Stone 2011), LWT (Atkeson, Moore, and Schaal 1997), HB (Wilson et al. 2007) in three dimensions, including the representations of transition and reward models, the types of transferred knowledge, and their performance. T and R denote the transition and the reward models. Gen. records how generalized a model representation is in each method: 'High' (generalization) means that a method could accurately represent all transition/reward models, whereas 'Low' means that not all models can be represented. Online, scalability, running time, jumpstart, accumulated reward, and asymptotic optimality are the performance characters. We relatively compare the methods on these characters. The optimality column presents the goal of a method: high reward or optimal policy. ✓: Yes, ×: No, –: Not applicable (or not a main focus of a method).

Method	**Representation**			**Transf. Know.**		**Performance**					
	T	R	Gen.	T	R	Online	Scal.	Time	Jumpst.	Acc. rew.	Optima.
RL-DT	✓	–	High	–	–	✓	×	Slow	–	Low	Pol.
Met-R.	✓	–	High	–	–	×	×	Slow	–	Low	Pol.
LSE-R.	✓	–	High	–	–	×	×	Slow	–	Low	Pol.
LWT	✓	–	Low	✓	–	✓	×	Fast	Low	High	Rew.
HB	–	✓	High	–	✓	✓	×	Slow	High	High	Pol.
loreRL	✓	–	Low	–	–	✓	✓	Fast	Low	High	Rew.
TES	✓	✓	Low	✓	✓	✓	✓	Fast	High	High	Rew.

However, there are still some questions as well as technical issues which we could not address in this dissertation. We next examine potential future studies that may further enhance the current theoretical framework.

8.2 Future work

We have discussed the situation where the world will be changed only when our agent takes an action. In general there may be other agencies changing the world, and the next state cannot easily be expressed as a combination of a small set of effects and a deterministic next state function. Adjustments to the CMDP theory are needed to accommodate such dynamic and non-stationary worlds.

We have also restricted our discussion to the situation where features in the states are fixed and known. In general those features may change in time following some distributions. In that case we may need to resort to stochastic features. Although our CMDP formalism allows that, empirical analysis needs to be conducted.

An input to our system is the whole state space of an environment. However, that prior knowledge may not be available in many applications. Further, *loreRL*, and *TES* have used value iteration for policy learning, which incurs costly computation. A more clever interleaving of model-building and policy formulation could be designed. For example, one could simply replace value iteration by *Dyna-Q* or *Prioritized Sweeping* algorithms. The decision to spend time in one of these tasks could also be learned by reinforcement.

We have limited ourselves to the cases of discrete state spaces and discrete actions, although the formalisms of *loreRL* and *TES* could be further generalized to continuous or hybrid action or state spaces. Actions and action effects could well be described by multiple features as long as there is an η-function that maps (state, effect)-pairs to the next states. More versatile state representations could also be implemented by views belonging to different model families.

Appendix A

Proof of theorem 1

Theorem 1 (Update Rule)**.** *Given $h(W) = \frac{1}{2}\|W\|_2^2$, a $K \times d$ average gradient matrix $\bar{G}^t$, and a regularization parameter $\lambda > 0$, the optimal solution of (4.2) is achieved column-wise as follows*

$$W_{\cdot i}^{t+1} = \begin{cases} \overrightarrow{0} & \text{if} \quad \|\bar{G}_{\cdot i}^t\|_2 \leq \lambda\sqrt{K}, \\ \frac{t}{\beta_t}\left(\frac{\lambda\sqrt{K}}{\|\bar{G}_{\cdot i}^t\|_2} - 1\right)\bar{G}_{\cdot i}^t & \text{otherwise.} \end{cases}$$

Proof. Let us rewrite the minimization problem,

$$W^{t+1} \quad = \quad \arg\min_W (\left\langle \bar{G}^t, W \right\rangle + \lambda\sqrt{K}\sum_i^d \|W_{\cdot i}\|_2 + \frac{\beta_t}{2t}\sum_i^d \|W_{\cdot i}\|_2^2)$$

Since the minimization problem is component-wise on one column of W, we can focus on each of the column of W separately to find its solution.

$$W_{\cdot i}^{t+1} = \arg\min_{W_{\cdot i}} \left(\left\langle \bar{G}_{\cdot i}^t, W_{\cdot i} \right\rangle + \lambda\sqrt{K}\|W_{\cdot i}\|_2 + \frac{\beta}{2t}\|W_{\cdot i}\|_2^2\right)$$

Since inner product of 2 vectors of same length will have smallest value when the 2 vectors are in opposite direction, solution to the above minimization problem should be $W_{\cdot i}^{t+1} = \varphi\bar{G}_{\cdot i}^t$ where $\varphi \leq 0$. We now need to solve the following minimization

problem,

$$\varphi \quad = \quad \arg\min_{\varphi \leq 0} \left(\varphi \|\bar{G}^t_{\cdot i}\|_2^2 - \varphi\lambda\sqrt{K}\|\bar{G}^t_{\cdot i}\|_2 + \varphi^2 \frac{\beta}{2t}\|\bar{G}^t_{\cdot i}\|_2^2 \right)$$

Solving for the minimum point of that familiar quadratic function, we have

$$\varphi = \begin{cases} \frac{t}{\beta_t}\left(\frac{\lambda\sqrt{K}}{\|\bar{G}^t_{\cdot i}\|_2} - 1\right) & \text{if } \|\bar{G}^t_{\cdot i}\|_2 > \lambda\sqrt{K}, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, the update rule is as in theorem 1. □

Appendix B

Proof of theorem 2

Theorem 2 (Regret Bound)**.** *Let the sequence of* $\{W^t\}_{t\geq 1}$ *be generated by the update rule (4.3), and assume that there exists a constant G such that* $\|G^t\|_*^2 \leq G^2, \forall t \geq 1$. *If we choose* $\beta_t = \alpha\sqrt{t}$ *where* $\alpha > 0$*, then for any* $t \geq 1$ *and for any W that satisfies* $h(W) \leq D^2$ *where D is a constant, the average regret is bounded as*

$$\frac{R_t(W)}{t} \leq \frac{\Delta}{\sqrt{t}}, \quad t = 1, 2, 3.., \tag{B.1}$$

where $\Delta = \left(\alpha D^2 + \frac{G^2}{\alpha}\right)$.

The proof of this theorem closely follows Xiao's proof (see Appendix B in (Xiao 2009)) for the logistic regression function. However, we reproduce it here for completeness.

Let S^t denote the sum of the subgradients obtained up to time t in the *mDAGL* method, that is,

$$S^t = \sum_{\tau=1}^{t} G^\tau = t\bar{G}^\tau, \tag{B.2}$$

with the initialization $S^0 = 0$. The the Equation 4.2 in *mDAGL* algorithm is equivalent to

$$W^{t+1} = \arg\min_W \{\langle S^t, W\rangle + t\Psi(W) + \beta_t h(W)\}. \tag{B.3}$$

In order to prove the regret bound, we need four Lemmas 9, 10, 11, and 12

in Xiao's proof. Those results directly extend to the case of multinomial logistic regression, so we use them without providing proofs.

Let W^0 be the unique minimizer of $h(W) = \frac{1}{2}\|W\|_2^2$, e.g.

$$W^0 = \arg\min_W h(W) \in \arg\min_W \Psi(W).$$

Let $\{\beta_t\}_{t\geq 1} = \alpha\sqrt{t}$, and $\beta_0 = \beta_1$. For each $t \geq 0$, we define two conjugate-type functions:

$$U^t(S) = \max_{W\in F_D} \left\{\left\langle S, W - W^0\right\rangle - t\Psi(W)\right\}, \tag{B.4}$$

$$V^t(S) = \max_W \left\{\left\langle S, W - W^0\right\rangle - t\Psi(W) - \beta_t h(W)\right\}, \tag{B.5}$$

where $F_D = \{W \in dom\Psi | h(W) \leq D^2\}$. The maximum in B.4 is always achieved because F_D is a nonempty compact set (which always contains W^0. For all $t \geq 0$, we have that all the functions $t\Psi(W) + \beta_t h(W)$ are strongly convex. Therefore, the maximum in B.5 is always achieved, and the maximizer is unique.

Let E be a finite-dimensional real vector space, endowed with a norm $\|\cdot\|$. Let E^* be the vector space of all linear functions on E. The dual space E^* is endowed with the dual norm $\|\cdot\|_*$. However, we will represent vectors in these spaces as matrices corresponding to W. We then have $domU^t = domV^t = E^*$ for all $t \geq 0$. Moreover, by the assumption $\Psi(W^0) = h(W^0) = 0$, both the functions are nonnegative.

Lemma 9. *For any $S \in E^*$ and $t \geq 0$, we have*

$$U^t(S) \leq V^t(S) + \beta_t D^2.$$

Let $z^t(S)$ denote the unique maximizer in the definition of $V^t(S)$; in other words,

$$\begin{aligned} z^t(S) &= \arg\max_W \left\{\left\langle S, W - W^0\right\rangle - r\Psi(W) - \beta_t h(W)\right\} \\ &= \arg\min_W \left\{\langle -S, W\rangle + t\Psi(W) + \beta_t h(W)\right\}. \end{aligned}$$

Comparing with the Equation B.3, we have

$$W^{t+1} = z^t(-S^t), \quad \forall t \geq 0.$$

Lemma 10. *The function V^t is convex and differentiable. Its gradient is given by*

$$\nabla V^t(S^t) = z^t(S) - W^0. \tag{B.6}$$

Moreover, the gradient is Lipschitz continuous with constant $1/\beta_t$; that is

$$\|\nabla V^t(S^1) - \nabla V^t(S^2)\| \leq \frac{1}{\beta_t}\|S^1 - S^2\|_*, \quad \forall S^1, S^2 \in E^*.$$

A direct consequence of Lemma 10 is the following inequality:

$$V^t(S + G) \leq V^t(S) + \langle G, \nabla V^t(S)\rangle + \frac{1}{2\beta_t}\|G\|_*^2, \quad \forall S, G \in E^*. \tag{B.7}$$

Lemma 11. *For each $t \geq 1$, we have*

$$V^t(-S^t) + \Psi(W^{t+1}) \leq V^{t-1}(-S^t) + (\beta_{t-1} - \beta_t)h(W^{t+1}).$$

Since by assumption $h(W^{t+1}) \geq 0$ and the sequence $\{\beta_t\}_{t\geq 1}$ is nondecreasing, we have

$$V^t(-S^t) + \Psi(W^{t+1}) \leq V^{t-1}(-S^t), \quad \forall t \geq 2. \tag{B.8}$$

For $t = 1$, Lemma 11 gives

$$V^1(-S^1) + \Psi(W^2) \leq V^0(-S^1) + (\beta_0 - \beta_1)h(W^2). \tag{B.9}$$

Lemma 12. *Provided that $\beta_1 = \alpha > 0$, and $h(W) = \frac{1}{2}\|W\|_2^2$, we have*

$$h(W^2) \leq \frac{2\|G^1\|_*^2}{\beta_1^2}. \tag{B.10}$$

Bounding the Regret

To measure the quality of the solutions $W^1, ..., W^t$, we define the following gap sequence:

$$\delta_t = \max_{W \in F_D} \left\{ \sum_{\tau=1}^{t} (\langle G^\tau, W^\tau - W \rangle + \Psi(W^\tau)) - t\Psi(W) \right\}, \quad t = 1, 2, 3, ... \tag{B.11}$$

The gap δ_t is an upper bound on the regret $R_t(W), \forall W \in F_D$. To see this, we use the assumption $W \in F_D$ and convexity of the item-wise loss function $l_t(W)$ in the following:

$$\begin{aligned} \delta_t &\geq \sum_{\tau=1}^{t} (\langle G^\tau, W^\tau - W \rangle + \Psi(W^\tau)) - t\Psi(W) \\ &\geq \sum_{\tau=1}^{t} (l_\tau(W^\tau) - l_\tau(W) + \Psi(W^\tau)) - t\Psi(W) \\ &= \sum_{\tau=1}^{t} (l_\tau(W^\tau) + \Psi(W^\tau)) - \sum_{\tau=1}^{t} (l_\tau(W) + \Psi(W)) = R_t(W). \end{aligned} \tag{B.12}$$

Now we derive the upper bound on δ_t. Adding and subtracting the sum $\sum_{\tau=1}^{t} \left\langle G^\tau, W^0 \right\rangle$ in the definition B.11, we have

$$\delta_t = \sum_{\tau=1}^{t} \left(\left\langle G^\tau, W^\tau - W^0 \right\rangle + \Psi(W^\tau) \right) + \max_{W \in F_D} \left\{ \left\langle S^t, W^0 - W \right\rangle - t\Psi(W) \right\}. \tag{B.13}$$

We observe that the maximization term in B.13 is in fact $U^t(-S^t)$. Therefore, by applying Lemma 9, we have

$$\delta_t \leq \sum_{\tau=1}^{t} \left(\left\langle G^\tau, W^\tau - W^0 \right\rangle + \Psi(W^\tau) \right) + V^t(-S^t) + \beta_t D^2. \tag{B.14}$$

Next, we show that Δ_t defined in B.1 is an upper bound for the right-hand side of the inequality B.14. For any $\tau \geq 2$, we have

$$
\begin{aligned}
V_\tau(-S^\tau) + \Psi(W^{\tau+1} &\leq V^{\tau-1}(-S^\tau) \\
&= V^{\tau-1}(-S^{\tau-1} - G^\tau) \\
&\leq V^{\tau-1}(-S^{\tau-1}) + \left\langle -G^\tau, \nabla V^{\tau-1}(-S^{\tau-1}) \right\rangle + \frac{\|G^\tau\|_*^2}{2\beta_{\tau-1}} \\
&= V^{\tau-1}(-S^{\tau-1}) + \left\langle -G^\tau, W^\tau - W^0 \right\rangle + \frac{\|G^\tau\|_*^2}{2\beta_{\tau-1}}
\end{aligned}
$$

where the four steps above used B.8,B.2,B.7,B.6, respectively. Therefore,

$$
\left\langle G^\tau, W^\tau - W^0 \right\rangle + \Psi(W^{\tau+1}) \leq V^{\tau-1}(-S^{\tau-1}) - V^\tau(-S^\tau) + \frac{\|G^\tau\|_*^2}{2\beta_{\tau-1}}, \quad \forall \tau \geq 2.
$$

For $\tau = 1$, we have a similar inequality

$$
\left\langle G^1, W^1 - W^0 \right\rangle + \Psi(W^2) \leq V^0(-S^0) - V^1(-S^1) + \frac{\|G^1\|_*^2}{2\beta_0} + (\beta_0 - \beta_1)h(W^2),
$$

where the additional term $(\beta_0 - \beta_1)h(W^2)$ comes from using (B.9). Summing the above inequalities for $\tau = 1, ..., t$, and noting that $V^0(-S^0) = V^0 = 0$, we arrive at

$$
\sum_{\tau=1}^{t} \left(\left\langle G^\tau, W^\tau - W^0 \right\rangle + \Psi(W^{\tau+1}) \right) + V^t(-S^t) \leq (\beta_0 - \beta_1)h(W^2) + \frac{1}{2} \sum_{\tau=1}^{t} \frac{\|G^\tau\|_*^2}{\beta_{\tau-1}}.
$$

Using $W^1 = W^0 \in \arg\min_W \Psi(W)$, we have $\Psi(W^{t+1}) \geq \Psi(W^0) = \Psi(W^1)$. Therefore, adding the nonpositive quantity $\Psi(W^1) - \Psi(W^{\tau+1})$ to the left-hand side of the above inequality yields

$$
\sum_{\tau=1}^{t} \left(\left\langle G^\tau, W^\tau - W^0 \right\rangle + \Psi(W^\tau) \right) + V^t(-S^t) \leq (\beta_0 - \beta_1)h(W^2) + \frac{1}{2} \sum_{\tau=1}^{t} \frac{\|G^\tau\|_*^2}{\beta_{\tau-1}}. \quad \text{(B.15)}
$$

Combining the inequalities B.12,B.14,B.15, and using Lemma 12, and the assump-

tion $\beta_0 = \beta_1$,

$$R_t(W) \leq \delta_t \quad \leq \quad \beta_t D^2 + \frac{1}{2} \sum_{\tau=1}^{t} \frac{\|G^\tau\|_*^2}{\beta_{\tau-1}}.$$

Using the assumption that $\|G^t\|_*^2 \leq G, \ \forall t > 0$, where G is a constant, we have

$$\begin{aligned} R_t(W) &\leq \beta_t D^2 + \frac{G^2}{2} \sum_{\tau=0}^{t-1} \frac{1}{\beta_\tau} \\ &= \alpha \sqrt{t} D^2 + \frac{G^2}{2\alpha} \left(1 + \sum_{\tau=1}^{t-1} \frac{1}{\sqrt{\tau}} \right). \end{aligned}$$

Next using the inequality

$$\sum_{\tau=1}^{t-1} \frac{1}{\sqrt{\tau}} \leq 1 + \int_1^t \frac{1}{\sqrt{\tau}} d\tau = 2\sqrt{t} - 1,$$

we get

$$R_t(W) \leq \alpha \sqrt{t} D^2 + \frac{G^2}{2\alpha}(1 + (2\sqrt{t} - 1)) = \left(\alpha D^2 + \frac{G^2}{\alpha} \right) \sqrt{t}.$$

This proves the regret bound in Theorem 2.

Appendix C

Proof of theorem 3

This proof was written by Zhuoru Li. The proof has also been reported in our joint work (Nguyen et al. 2013).

Theorem 3 (Difference in Value Function)**.** *Let* $M_1 = (S, f, A, T(W^{M_1}), E, R, \gamma)$ *and* $M_2 = (S, f, A, T(W^{M_2}), E, R, \gamma)$ *be two CMDPs with optimal policies* π_1 *and* π_2 *respectively. Let us denote by* V_π^M *the value function for policy* π *in CMDP* M*. Let*

$$\epsilon_1 = 2\sqrt{\max_{a\in A, e\in E} \|W_e^{(a),M_1} - W_e^{(a),M_2}\|_1 \sup_s \|x(s)\|_1},$$

then $\max_{s\in S}\left(V_{\pi_1}^{M_2}(s) - V_{\pi_2}^{M_2}(s)\right) \leq \frac{2\gamma V_{\max}\epsilon_1}{1-\gamma}$,

where $W_e^{(a),M_1}$ *and* $W_e^{(a),M_2}$ *refer to the vector of coefficients corresponding to class* $E = e$ *under action* a *in model* M_1 *and* M_2 *respectively,* $\|\cdot\|_1$ *is the 1-norm of vector, and* V_{max} *is the maximum value of any state for any policy in either of the CMDPs.*

Proof. For any action a, consider the following expression, where x is a vector of all state attributes and features extracted from state s and e is the action effect leading to s' from s.

$$\begin{aligned}
&\sum_{s'\in S} P^{M_1}(s'|s)\log\left(\frac{P^{M_1}(s'|s)}{P^{M_2}(s'|s)}\right)\\
=&\sum_{e\in E} P^{M_1}(e|s)\log\left(\frac{P^{M_1}(e|s)}{P^{M_2}(e|s)}\right)
\end{aligned}$$

$$
\begin{aligned}
&\leq \max_{e\in E} \log\left(\frac{P^{M_1}(e|s)}{P^{M_2}(e|s)}\right) \\
&= \max_{e\in E} \log\left(\frac{\exp(W_e^{M_1}x)/\sum_{e'\in E}\exp(W_{e'}^{M_1}x)}{\exp(W_e^{M_2}x)/\sum_{e'\in E}\exp(W_{e'}^{M_2}x)}\right) \\
&= \max_{e\in E}\left[\log\left(\frac{\exp(W_e^{M_1}x)}{\exp(W_e^{M_2}x)}\right) - \log\left(\frac{\sum_{e'\in E}\exp(W_{e'}^{M_1}x)}{\sum_{e'\in E}\exp(W_{e'}^{M_2}x)}\right)\right] \\
&= \max_{e\in E}\left[(W_e^{M_1} - W_e^{M_2})x - \log\left(\frac{\sum_{e'\in E}\exp(W_{e'}^{M_1}x)}{\sum_{e'\in E}\exp(W_{e'}^{M_2}x)}\right)\right] \\
&\leq \max_{e\in E}\left[(W_e^{M_1} - W_e^{M_2})x - \log\left(\min_{e'\in E}\left(\frac{\exp(W_{e'}^{M_1}x)}{\exp(W_{e'}^{M_2}x)}\right)\right)\right] \\
&= \max_{e\in E}\left[(W_e^{M_1} - W_e^{M_2})x - \min_{e'\in E}\left(\log\left(\frac{\exp(W_{e'}^{M_1}x)}{\exp(W_{e'}^{M_2}x)}\right)\right)\right] \\
&= \max_{e\in E}\left[(W_e^{M_1} - W_e^{M_2})x - \min_{e'\in E}\left((W_{e'}^{M_1} - W_{e'}^{M_2})x\right)\right] \\
&\leq \max_{e\in E}[\|W_e^{M_1} - W_e^{M_2}\|_1 \sup_s \|x(s)\|_1 + \max_{e'\in E}(\|W_{e'}^{M_1} - W_{e'}^{M_2}\|_1 \sup_s \|x(s)\|_1)] \\
&\leq 2\max_{e\in E}(\|W_e^{M_1} - W_e^{M_2}\|_1 \sup_s \|x(s)\|_1)
\end{aligned}
$$

The first step is from definition of *effect*. The second step is from the fact that weighted average of elements must be smaller than the largest one. The sixth step is from the property that if a_i and b_i are non-negative, then $(\sum_i a_i) / (\sum_i b_i) \geq \min_i(a_i/b_i)$. The seventh step is from monotonicity of logarithmic function.

By Pinsker's inequality,

$$
\begin{aligned}
&\sum_{s'\in S} P^{M_1}(s'|s)\log\left(\frac{P^{M_1}(s'|s)}{P^{M_2}(s'|s)}\right) \\
\geq &\frac{1}{2}(\sum_{s'\in S}|P^{M_1}(s'|s) - P^{M_2}(s'|s)|)^2
\end{aligned}
$$

which implies

$$
\begin{aligned}
&\sum_{s'\in S}|P^{M_1}(s'|s) - P^{M_2}(s'|s)| \\
\leq &2\sqrt{\max_{e\in E}(\|W_e^{M_1} - W_e^{M_2}\|_1 \sup_s \|x(s)\|_1)}
\end{aligned}
$$

Extending to all actions,

$$\begin{aligned}
&\sum_{s' \in S} |P^{M_1}(s'|s) - P^{M_2}(s'|s)| \\
\leq &\max_{a \in A} \left(2 \sqrt{\max_{e \in E} (\|W_e^{(a),M_1} - W_e^{(a),M_2}\|_1 \sup_s \|x(s)\|_1)} \right) \\
= &2 \sqrt{\max_{a \in A, e \in E} (\|W_e^{(a),M_1} - W_e^{(a),M_2}\|_1 \sup_s \|x(s)\|_1)}
\end{aligned}$$

To complete the theorem, the following lemma (see lemma 33 in (Li 2009)) is used without proof.

Lemma 1. *Let $M_1 = (S, A, P^{M_1}, R), M_2 = (S, A, P^{M_2}, R)$ be two MDPs, and fixed discount factor γ. π_1 and π_2 are their optimal policies respectively. Let V_π^M be the value function of π in MDP M. If*

$$\sum_{s' \in S} |P^{M_1} - P^{M_2}|(s'|s, a) \leq \epsilon$$

for every state-action (s, a), then $|V_{\pi_2}^{M_1}(s) - V_{\pi_2}^{M_2}(s)| \leq \frac{\gamma V_{max} \epsilon}{1-\gamma}$ and $|V_{\pi_1}^{M_2}(s) - V_{\pi_1}^{M_1}(s)| \leq \frac{\gamma V_{max} \epsilon}{1-\gamma}$, for every $s \in S$.

It is clear that

$$\begin{aligned}
&\max_{s \in S} \left(V_{\pi_2}^{M_2} - V_{\pi_1}^{M_2} \right) \\
= &\max_{s \in S} \left(V_{\pi_2}^{M_2} - V_{\pi_1}^{M_1} + V_{\pi_1}^{M_1} - V_{\pi_1}^{M_2} \right) \\
\leq &\max_{s \in S} \left(V_{\pi_2}^{M_2} - V_{\pi_2}^{M_1} + V_{\pi_1}^{M_1} - V_{\pi_1}^{M_2} \right) \\
\leq &\max_{s \in S} |V_{\pi_2}^{M_2} - V_{\pi_2}^{M_1}| + \max_{s \in S} |V_{\pi_1}^{M_1} - V_{\pi_1}^{M_2}| \\
\leq &\frac{2\gamma V_{max} \epsilon}{1-\gamma}.
\end{aligned}$$

The proof is therefore complete. □

Appendix D

Multinomial logistic regression functions

We list the W matrices used in the four different sets of multinomial logistic regression functions to generate the effect distributions of four actions, namely: move up, move left, move down, and move right. Each action may have its effect distribution determined by one of the four functions. The first set was used in the experiments in Chapter 5. The last three ones were for the experiments in Chapter 6.

The columns of the matrix correspond to the 9 indicator variables and a bias factor (brick, sand, soil, water, grass, wall-up, wall-left, wall-bottom, wall-right, bias) and rows correspond to possible effects for movements (up, left, down, right, not moved).

D.1 Set No.1 of logistic regression functions

D.1.1 Move up

$$W^{(1)} = \begin{pmatrix} 3.99 & 3.00 & 3.50 & 2.60 & 0.00 & -4.00 & 0.00 & 0.00 & 0.01 & 0.00 \\ 1.23 & 1.10 & 1.15 & 1.20 & 0.00 & 0.00 & -4.00 & 0.02 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 \\ 1.23 & 1.10 & 1.15 & 1.20 & 0.01 & 0.00 & 0.00 & 0.00 & -4.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 4.00 & 0.90 & 0.01 & 0.91 & 0.00 \end{pmatrix}$$

D.1.2 Move left

$$W^{(1)} = \begin{pmatrix} 1.23 & 1.10 & 1.15 & 1.20 & 0.00 & -4.00 & 0.00 & 0.01 & 0.00 & 0.01 \\ 3.99 & 3.00 & 3.50 & 2.60 & 0.00 & 0.00 & -4.00 & 0.00 & 0.00 & 0.00 \\ 1.23 & 1.10 & 1.15 & 1.20 & 0.00 & 0.00 & 0.00 & -4.00 & 0.02 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.90 & 4.00 & 0.90 & 0.00 & 0.01 \end{pmatrix}$$

D.1.3 Move down

$$W^{(1)} = \begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 \\ 1.23 & 1.10 & 1.15 & 1.20 & 0.00 & 0.01 & -4.00 & 0.01 & 0.00 & 0.02 \\ 3.99 & 3.00 & 3.50 & 2.60 & 0.00 & 0.00 & 0.00 & -4.00 & 0.00 & 0.00 \\ 1.23 & 1.10 & 1.15 & 1.20 & 0.01 & 0.0 & 0.00 & 0.00 & -4.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.90 & 4.00 & 0.90 & 0.00 \end{pmatrix}$$

D.1.4 Move right

$$W^{(1)} = \begin{pmatrix} 1.23 & 1.10 & 1.15 & 1.20 & 0.00 & -4.00 & 0.00 & 0.01 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.00 & 0.01 \\ 1.23 & 1.10 & 1.15 & 1.20 & 0.01 & 0.00 & 0.00 & -4.00 & 0.00 & 0.00 \\ 3.99 & 3.00 & 3.50 & 2.60 & 0.00 & 0.00 & 0.01 & 0.00 & -4.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.90 & 0.00 & 0.90 & 4.00 & 0.02 \end{pmatrix}$$

D.2 Set No.2 of logistic regression functions

D.2.1 Move up

$$W^{(2)} = \begin{pmatrix} 3.99 & 3.00 & 3.50 & 2.50 & 0.00 & -4.00 & 0.00 & 0.02 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.00 & -4.00 & 0.00 & 0.00 & 0.02 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.00 & 0.00 & 0.01 & -4.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 4.00 & 0.90 & 0.00 & 0.90 & 0.00 \end{pmatrix}$$

D.2.2 Move left

$$W^{(2)} = \begin{pmatrix} 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & -4.00 & 0.00 & 0.01 & 0.00 & 0.00 \\ 3.99 & 3.00 & 3.50 & 2.50 & 0.01 & 0.00 & -4.00 & 0.00 & 0.01 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.00 & 0.00 & -4.00 & 0.00 & 0.03 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.90 & 4.00 & 0.90 & 0.00 & 0.00 \end{pmatrix}$$

D.2.3 Move down

$$W^{(2)} = \begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.01 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.00 & -4.00 & 0.01 & 0.00 & 0.01 \\ 3.99 & 3.00 & 3.50 & 2.50 & 0.01 & 0.00 & 0.00 & -4.00 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.01 & 0.00 & 0.00 & -4.00 & 0.02 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.90 & 4.00 & 0.90 & 0.00 \end{pmatrix}$$

D.2.4 Move right

$$W^{(2)} = \begin{pmatrix} 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & -4.00 & 0.03 & 0.01 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.00 & 0.01 & -4.00 & 0.00 & 0.00 \\ 3.99 & 3.00 & 3.50 & 2.50 & 0.00 & 0.00 & 0.00 & 0.00 & -4.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.90 & 0.00 & 0.90 & 4.00 & 0.00 \end{pmatrix}$$

D.3 Set No.3 of logistic regression functions

D.3.1 Move up

$$W^{(3)} = \begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.01 & -4.00 & 0.00 & 0.00 & 0.01 \\ 3.99 & 3.00 & 3.50 & 2.50 & 0.00 & 0.00 & 0.00 & -4.01 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.01 & 0.00 & 0.00 & -4.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.90 & 4.00 & 0.93 & 0.01 \end{pmatrix}$$

D.3.2 Move left

$$W^{(3)} = \begin{pmatrix} 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & -4.00 & 0.01 & 0.01 & 0.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.00 & 0.00 & -4.00 & 0.00 & 0.00 \\ 3.99 & 3.00 & 3.50 & 2.50 & 0.01 & 0.00 & 0.01 & 0.00 & -4.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.90 & 0.00 & 0.90 & 4.01 & 0.00 \end{pmatrix}$$

D.3.3 Move down

$$W^{(3)} = \begin{pmatrix} 3.99 & 3.00 & 3.50 & 2.50 & 0.01 & -4.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.00 & -4.00 & 0.01 & 0.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.01 & 0.00 & 0.02 & -4.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 4.00 & 0.93 & 0.00 & 0.90 & 0.00 \end{pmatrix}$$

D.3.4 Move right

$$W^{(3)} = \begin{pmatrix} 0.80 & 1.5 & 1.15 & 1.35 & 0.01 & -4.01 & 0.00 & 0.01 & 0.00 & 0.00 \\ 3.99 & 3.00 & 3.50 & 2.50 & 0.00 & 0.00 & -4.00 & 0.00 & 0.00 & 0.01 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.00 & 0.00 & -4.00 & 0.02 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.90 & 4.01 & 0.90 & 0.001 & 0.00 \end{pmatrix}$$

D.4 Set No.4 of logistic regression functions

D.4.1 Move up

$$W^{(4)} = \begin{pmatrix} 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & -4.00 & 0.00 & 0.01 & 0.00 & 0.01 \\ 3.99 & 3.00 & 3.50 & 2.50 & 0.00 & 0.00 & -4.00 & 0.00 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.03 & 0.00 & 0.00 & -4.00 & 0.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.02 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.90 & 4.00 & 0.90 & 0.00 & 0.00 \end{pmatrix}$$

D.4.2 Move left

$$W^{(4)} = \begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.02 & -4.01 & 0.01 & 0.00 & 0.01 \\ 3.99 & 3.00 & 3.50 & 2.50 & 0.00 & 0.00 & 0.00 & -4.00 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.01 & 0.01 & 0.00 & 0.00 & -4.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.90 & 4.00 & 0.90 & 0.00 \end{pmatrix}$$

D.4.3 Move down

$$W^{(4)} = \begin{pmatrix} 0.80 & 1.5 & 1.15 & 1.35 & 0.0 & -4.00 & 0.00 & 0.01 & 0.00 & 0.01 \\ 3.99 & 3.00 & 3.50 & 2.50 & 0.02 & 0.00 & -4.00 & 0.00 & 0.01 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.00 & 0.01 & 0.00 & -4.00 & 0.01 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.001 & 0.90 & 4.00 & 0.90 & 0.00 & 0.00 \end{pmatrix}$$

D.4.4 Move right

$$W^{(4)} = \begin{pmatrix} 3.99 & 3.00 & 3.50 & 2.50 & 0.00 & -4.00 & 0.00 & 0.00 & 0.00 & 0.03 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.01 & 0.01 & -4.01 & 0.01 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.80 & 1.5 & 1.15 & 1.35 & 0.02 & 0.00 & 0.00 & 0.01 & -4.00 & 0.01 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 4.00 & 0.91 & 0.00 & 0.90 & 0.00 \end{pmatrix}$$

Appendix E

Value iteration algorithm

Algorithm 8 Value iteration

Input: MDP(S, A, T, R, γ)
Output: V

Initialize V arbitrarily, e.g. $\forall s \in S, V(s) \leftarrow 0$
repeat
  $\Delta \leftarrow 0$
  for each state s **in** S **do**
    $oldV \leftarrow V(s)$
    for each action a **in** A **do**
      $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s')$
    end for
    $V(s) \leftarrow max_a Q(s,a)$
    if $|V(s) - oldV| > \Delta$ **then**
      $\Delta \leftarrow |V(s) - oldV|$
    end if
  end for
until $\Delta < \epsilon$

Given the transition and the reward models, a simple dynamic programming technique is typically used to learn an optimal policy via solving the Bellman equations (Equation 2.3). Algorithm 8 shows *Value iteration* (Bellman 1957; Bertsekas 1987), a popular dynamic programming approach to optimal policy learning. In value iteration algorithm, the values of all states are alternately updated according to equation 2.3 in every iteration. The algorithm needs to repeat infinitely the iterations to converge

exactly to the optimal value function V^*. However, an optimal policy is usually discovered long before the optimal value function is found. Thus, it is common in practice to stop the algorithm when the maximum difference Δ between two consecutive value functions is smaller than an epsilon value ϵ. If $\Delta < \epsilon$, then the value function by the Algorithm 8 and the optimal value function are not different by more than $2\epsilon/(1-\gamma)$ at any state (Williams and Baird 1993).

References

Atkeson, C. G.; Moore, A. W.; and Schaal, S. 1997. Locally weighted learning. *Journal of Artificial Intelligence Review* 11:11–73.

Bellman, R. 1957. *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press.

Bertsekas, D. P. 1987. *Dynamic programming: deterministic and stochastic models*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2001. Stochastic dynamic programming with factored representations. *Journal of Artificial Intelligence* 121.

Brafman, R. I., and Tennenholtz, M. 2002. R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3:213–231.

Celiberto, J. L. A.; Matsuura, J. P.; De Mantaras, R. L.; and Bianchi, R. A. C. 2011. Using cases as heuristics in reinforcement learning: a transfer learning application. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 2 of *IJCAI '11*, 1211–1217.

Chakraborty, D., and Stone, P. 2011. Structure learning in ergodic factored MDPs without knowledge of the transition function's in-degree. In *Proceedings of the International Conference on Machine Learning*, ICML '11.

Dawid, A. 1984. Statistical theory: The prequential approach. *Journal of the Royal Statistical Society A* 147:278–292.

Degris, T.; Sigaud, O.; and Wuillemin, P.-H. 2006. Learning the structure of factored Markov decision processes in reinforcement learning problems. In *Proceedings of the International Conference on Machine Learning*, ICML '06, 257–264.

Diuk, C.; Li, L.; and Leffler, B. R. 2009. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, ICML '09, 249–256.

Doya, K.; Samejima, K.; Katagiri, K.; and Kawato, M. 2002. Multiple model-based reinforcement learning. *Journal of Neural Computation* 14:1347–1369.

Fernández, F.; García, J.; and Veloso, M. 2010. Probabilistic policy reuse for inter-task transfer learning. *Journal of Robotics and Autonomous Systems* 58:866–871.

Hester, T., and Stone, P. 2009. Generalized model learning for reinforcement learning in factored domains. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, volume 2 of *AAMAS '09*, 717–724.

Hester, T., and Stone, P. 2012. TEXPLORE : real-time sample-efficient reinforcement learning for robots. *Machine Learning* 1–45.

Kaelbling, P. L.; Littman, M.; and Moore, A. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285.

Kearns, M., and Koller, D. 1999. Efficient reinforcement learning in factored MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 2 of *IJCAI '99*, 740–747.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Proceedings of the European Conference on Machine Learning*, ECML '06, 282–293.

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Konidaris, G., and Barto, A. 2009. Efficient skill learning using abstraction selection. In *Proceedings of the International Joint Conference on Artifical Intelligence*, IJCAI '09, 1107–1112.

Kroon, M., and Whiteson, S. 2009. Automatic feature selection for model-based reinforcement learning in factored MDPs. In *Proceedings of the International Conference on Machine Learning and Applications*, ICMLA '09.

Lee, S., and Wright, S. 2012. Manifold identification of dual averaging methods for regularized stochastic online learning. *Journal of Machine Learning Research*.

Leffler, B. R.; Littman, M. L.; and Edmunds, T. 2007. Efficient reinforcement learning with relocatable action models. In *Proceedings of the National Conference on Artificial Intelligence*, volume 1 of *AAAI '07*.

Li, L. 2009. *A unifying framework for computational reinforcement learning theory*. Ph.D. Dissertation, Rutgers, The State of University of New Jersey.

Maclin, R.; Shavlik, J.; Torrey, L.; Walker, T.; and Wild, E. 2005. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2 of *AAAI '05*, 819–824.

Madden, M. G., and Howley, T. 2004. Transfer of experience between reinforcement learning environments with progressive difficulty. *Journal of Artificial Intelligence Review* 21:375–398.

McCarthy, J. 1963. Situations, actions, and causal laws. Technical Report Memo 2, Stanford Artificial Intelligence Project, Stanford University.

Moore, A. W., and Atkeson, C. G. 1993. Prioritized Sweeping: Reinforcement learning with less data and less time. *Journal of Machine Learning* 13:103–130.

Nguyen, T. T.; Li, Z.; Silander, T.; and Leong, T.-Y. 2013. Online feature selection for model-based reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, ICML '13.

Nguyen, T. T.; Silander, T.; and Leong, T.-Y. 2012a. Transfer learning as representation selection. In *International Conference on Machine Learning Workshop on Representation Learning*.

Nguyen, T. T.; Silander, T.; and Leong, T.-Y. 2012b. Transferring expectations in model-based reinforcement learning. In *Proceedings of the Advances in Neural Information Processing Systems*, NIPS '12.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Ross, S., and Pineau, J. 2008. Model-based Bayesian reinforcement learning in large structured domains. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, UAI '08, 476–483.

Rummery, G. A., and Niranjan, M. 1994. Online Q-learning using connectionist systems. Technical Report CUEF/F-INFENG/TR 166, Cambridge University Engineering Department.

Savage, L. J. 1971. Elicitation of personal probabilities and expectations. *Journal of the American Statistical Association* 66(336):783–801.

Sharma, M.; Holmes, M.; Santamaria, J.; Irani, A.; Isbell, C.; and Ram, A. 2007. Transfer learning in real-time strategy games using hybrid cbr/rl. In *Proceedings of the International Joint Conference on Artifical Intelligence*, IJCAI '07, 1041–1046.

Sherstov, A. A., and Stone, P. 2005. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2 of *AAAI '05*, 1024–1029.

Silva, B. C. D.; Basso, E. W.; Bazzan, A. L. C.; and Engel, P. M. 2006. Dealing with non-stationary environments using context detection. In *Proceedings of the International Conference on Machine Learning*, ICML '06, 217–224.

Skinner, B. F. 1953. *Science and Human Behavior*. New York: The Free Press.

Soni, V., and Singh, S. 2006. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of the National Conference on Artificial Intelligence*, volume 1 of *AAAI '06*, 494–499.

Strehl, A. L., and Littman, M. L. 2007. Online linear regression and its application to model-based reinforcement learning. In *Proceedings of the Advances in Neural Information Processing Systems*, NIPS '07, 737–744.

Strehl, A. L.; Diuk, C.; and Littman, M. L. 2007. Efficient structure learning in factored-state MDPs. In *Proceedings of the National Conference on Artificial Intelligence*, volume 1 of *AAAI '07*, 645–650.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the International Conference on Machine Learning*, ICML '90, 216–224.

Tanaka, F., and Yamamura, M. 2003. Multitask reinforcement learning on the distribution of MDPs. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3 of *ICRA '03*, 1108 – 1113.

Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10:1633–1685.

Taylor, M. E.; Jong, N. K.; and Stone, P. 2008. Transferring instances for model-based reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, volume 5212 of *LNAI*.

Thorndike, E. L., and Woodworth, R. S. 1901. The influence of improvement in one mental function upon the efficiency of other functions. *Journal of Psychological Review* 8:247–261.

Van Seijen, H.; Bakker, B.; and Kester, L. 2008. Switching between different state representations in reinforcement learning. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications*, AIA '08, 226–231.

Walsh, T. J.; Szita, I.; Diuk, C.; and Littman, M. L. 2009. Exploring compact reinforcement-learning representations with linear regression. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, UAI '09.

Walsh, T. J.; Li, L.; and Littman, M. L. 2006. Transferring state abstractions between MDPs. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*.

Williams, R., and Baird, L. C. 1993. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-14, Northeastern University, College of Computer Science, Boston, MA.

Wilson, A.; Fern, A.; Ray, S.; and Tadepalli, P. 2007. Multi-task reinforcement learning: A hierarchical Bayesian approach. In *Proceedings of the International Conference on Machine Learning*, ICML '07.

Xiao, L. 2009. Dual averaging methods for regularized stochastic learning and online optimization. In *Proceedings of the Advances in Neural Information Processing Systems*, NIPS '09.

Yang, H.; Xu, Z.; King, I.; and Lyu, M. R. 2010. Online learning for group lasso. In *Proceedings of the International Conference on Machine Learning*, ICML '10.

Zhu, X.; Ghahramani, Z.; and Lafferty, J. 2005. Time-sensitive dirichlet process mixture models. Technical Report CMU-CALD-05-104, School of Computer Science, Carnegie Mellon University.